Application de la métaprogrammation template à la conception de bibliothèques actives de parallélisation assistée

Alexis Pereda

Thèse encadrée par Bruno Bachelet, David Hill, Claude Mazel

Université Clermont Auvergne

01/07/2021

Membres du jury

Alexandre Guitton	*Professeur des universités*	Université Clermont Auvergne	Président
Joël Falcou	*Maître de conférences HDR*	Université Paris-Saclay	Rapporteur
Françoise Baude	*Professeur des universités*	Université Côte d'Azur	Rapporteuse
Éric Innocenti	*Maître de conférences*	Université de Corse	Examinateur
Bruno Bachelet	*Maître de conférences HDR*	Université Clermont Auvergne	Directeur de thèse
David Hill	*Professeur des universités*	Université Clermont Auvergne	Co-directeur de thèse

Introduction

Performances $\propto$ nombre de cœurs

$\Longrightarrow$ besoin de bénéficier du parallélisme des cœurs

mais difficile donc sous-exploité

Problématique :

expertise parallélisation $\neq$ expertise métier

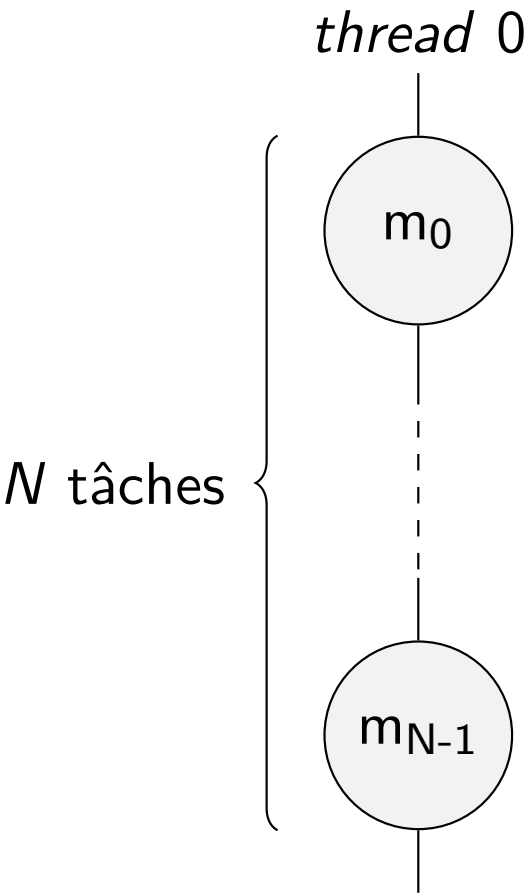
$\Longrightarrow$ besoin d'outils pour assister la parallélisation

Plan

Plan

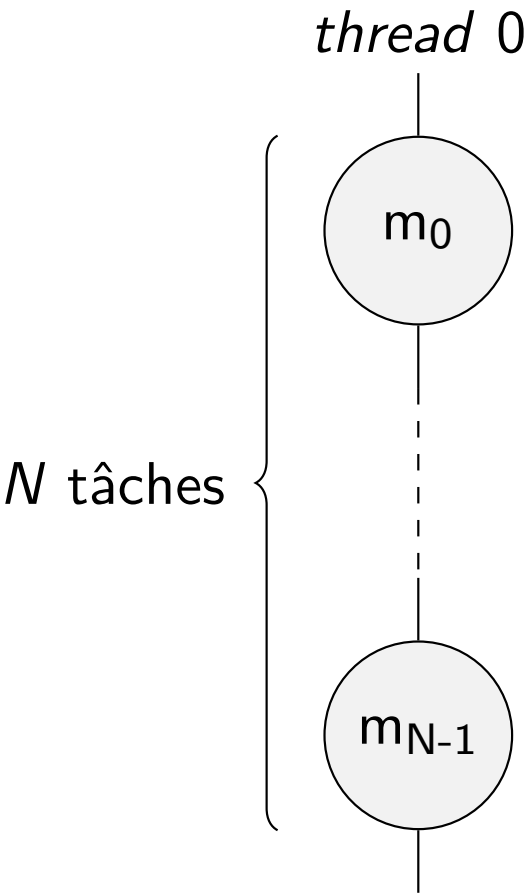
Idée générale

Exécution séquentielle



Idée générale

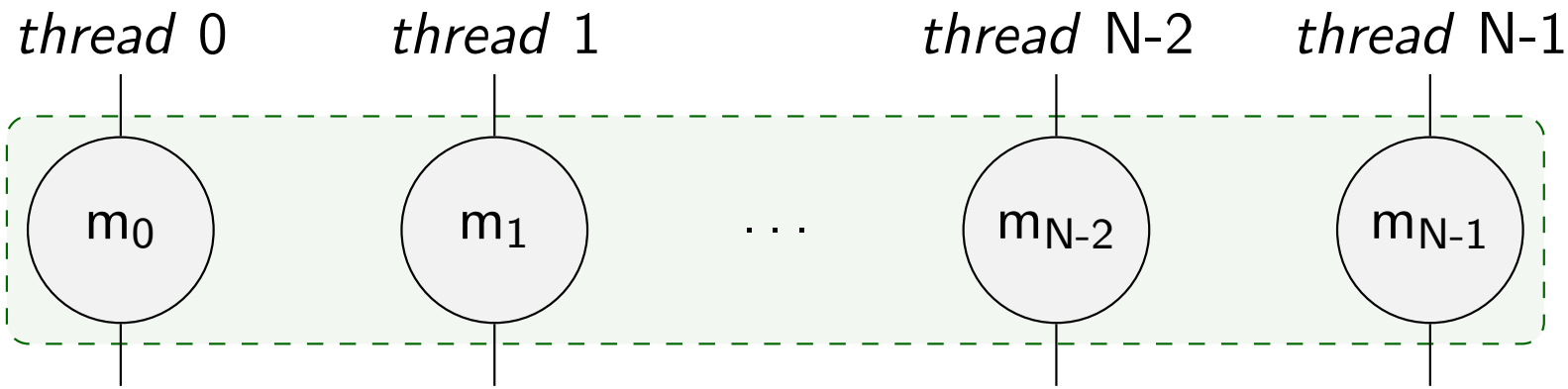
Exécution séquentielle



- matériel multicœur
- tâches $m_{\{0..N-1\}}$ indépendantes

Idée générale

Exécution parallèle idéale



Difficultés de la parallélisation

Difficultés et inconvénients

- déterminer les parties parallélisables
  - → analyse de dépendances

Difficultés de la parallélisation

Difficultés et inconvénients

- déterminer les parties parallélisables
  - → analyse de dépendances
- parallélisation correcte et maintenable
  - → composition de fonctions parallèles
  - → séparation des domaines d'expertise

Difficultés de la parallélisation

Difficultés et inconvénients

- déterminer les parties parallélisables
  - → analyse de dépendances
- parallélisation correcte et maintenable
  - → composition de fonctions parallèles
  - → séparation des domaines d'expertise
- besoin de synchronisation
  - → attente de la fin de tâches
  - → partage de ressources

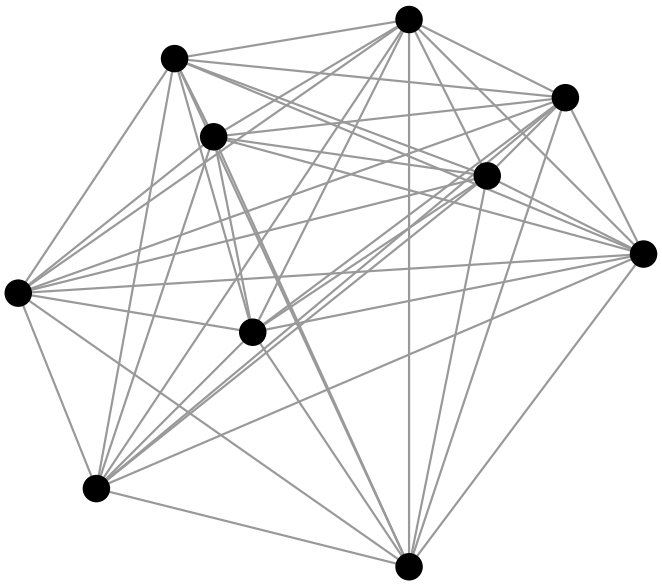
Difficultés de la parallélisation

Difficultés et inconvénients

- déterminer les parties parallélisables
  - → analyse de dépendances
- parallélisation correcte et maintenable
  - → composition de fonctions parallèles
  - → séparation des domaines d'expertise
- besoin de synchronisation
  - → attente de la fin de tâches
  - → partage de ressources
- non répétabilité des exécutions
  - → cas des nombres pseudo-aléatoires

Problème du voyageur de commerce

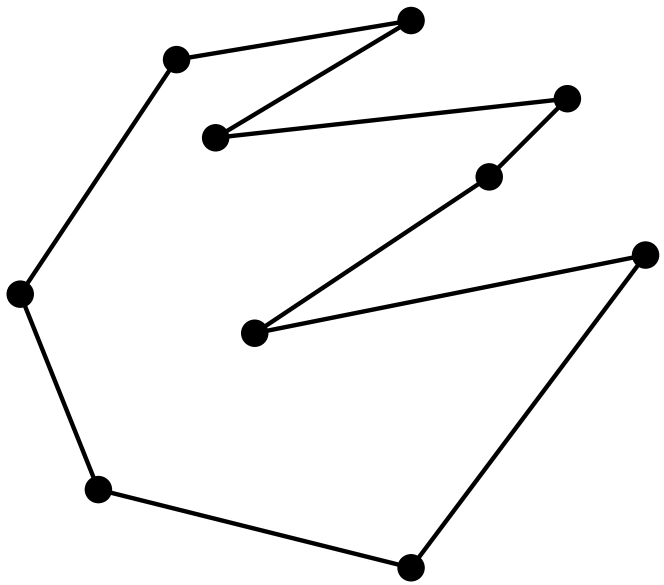
Instance



- Graphe $G = (V, A, c)$

Problème du voyageur de commerce

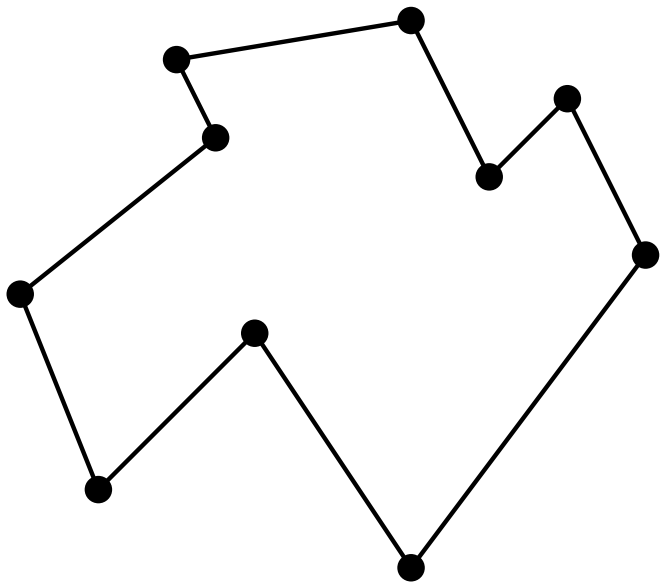
Solution quelconque



- Graphe $G = (V, A, c)$
- coût $c_s = \sum_{i=2}^{|V|} c((s[i-1], s[i]))$

Problème du voyageur de commerce

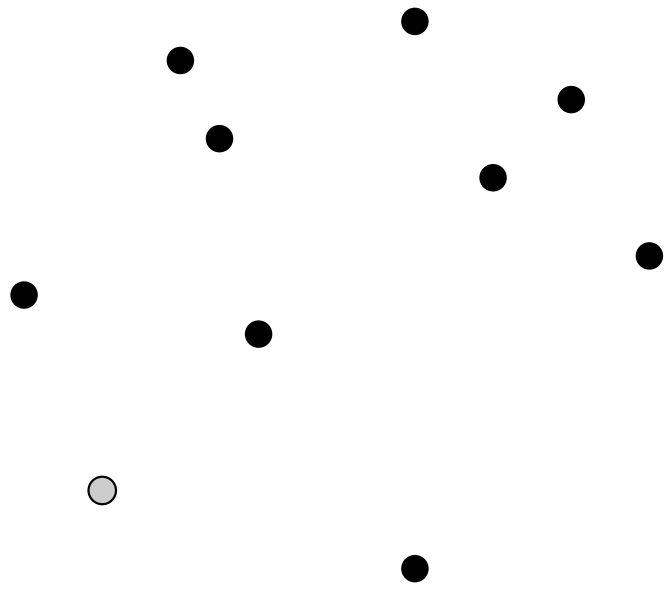
Solution optimale



- Graphe $G = (V, A, c)$
- coût $c_s = \sum_{i=2}^{|V|} c((s[i-1], s[i]))$
- Problème d'optimisation
- NP-difficile

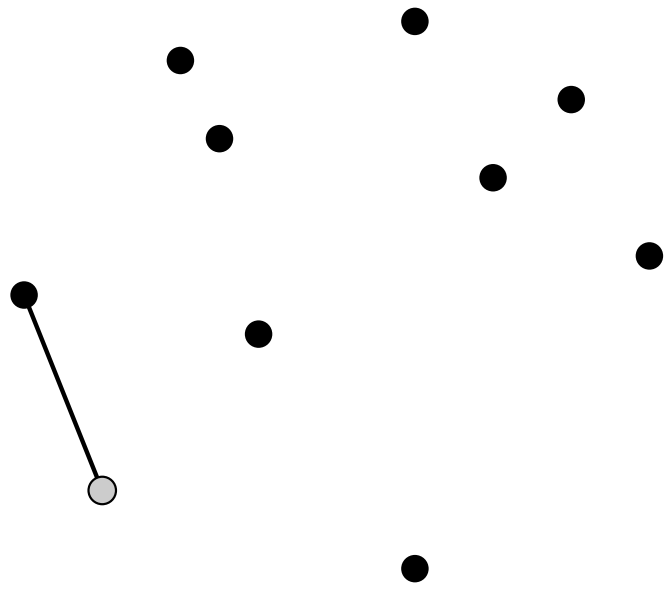
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



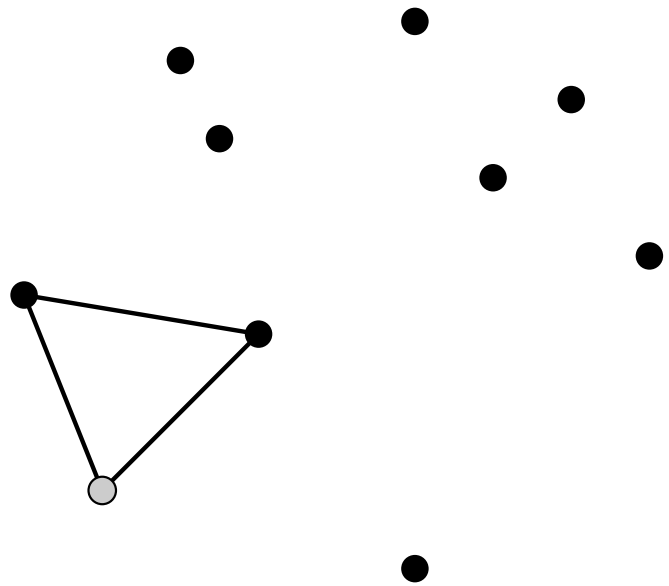
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



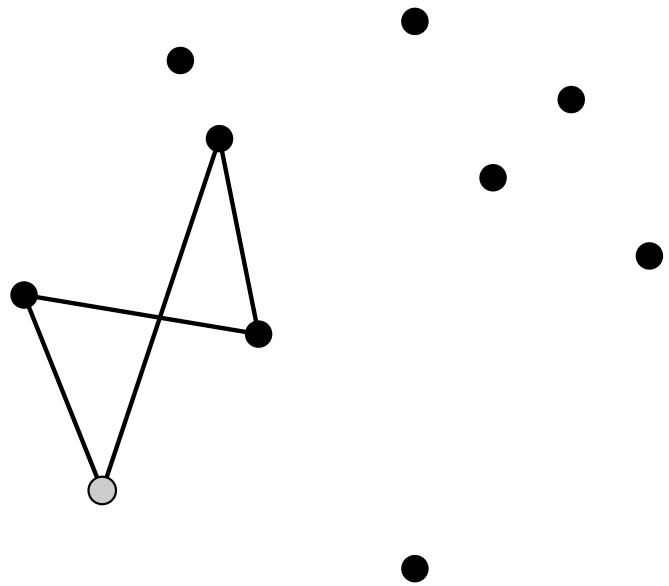
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



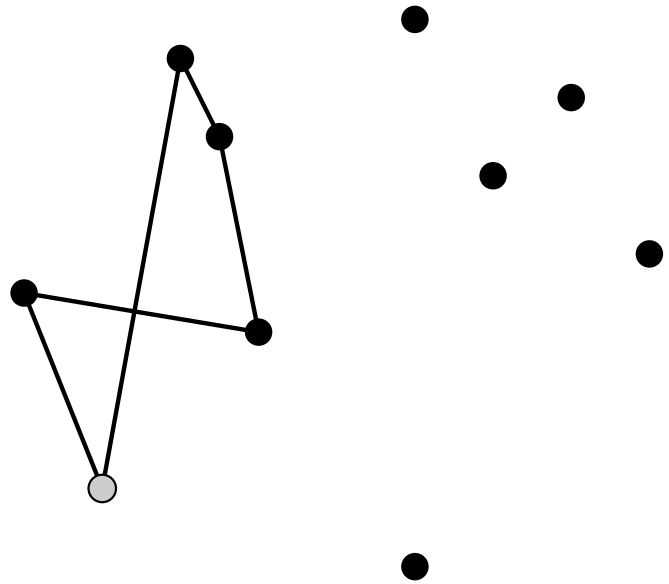
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



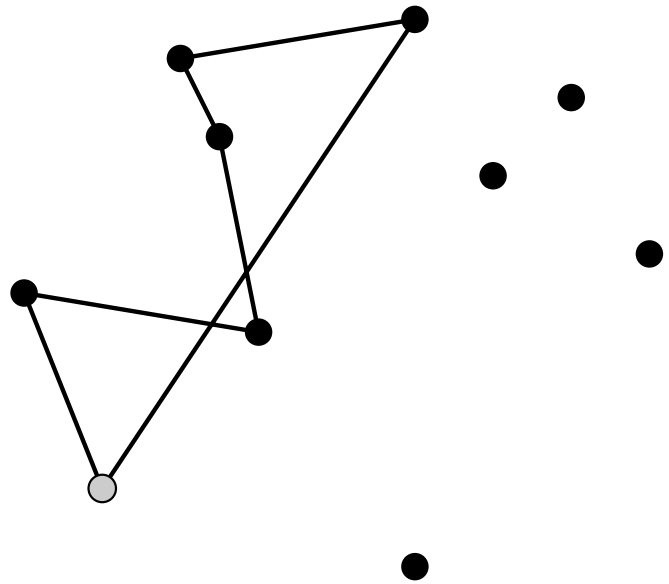
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



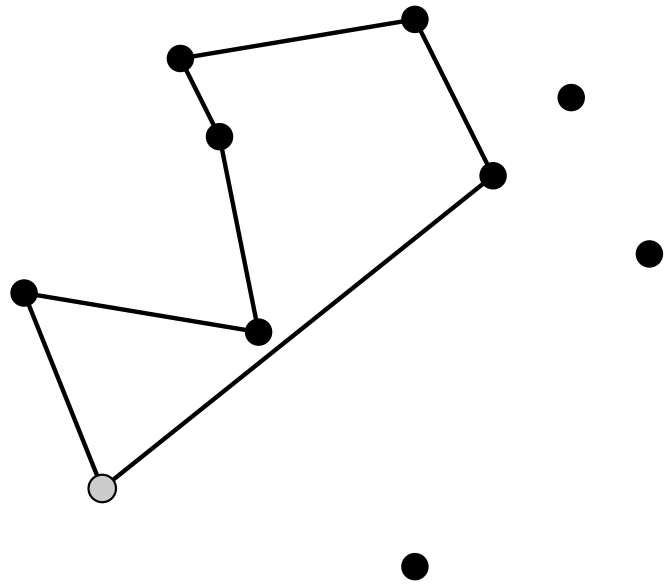
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



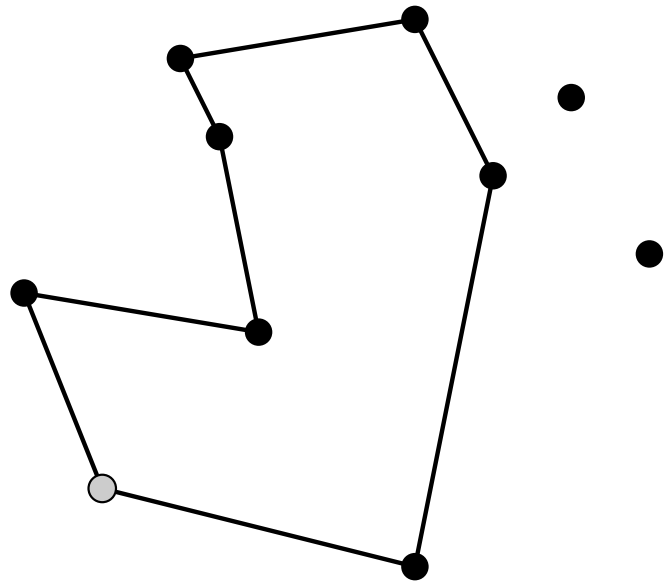
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



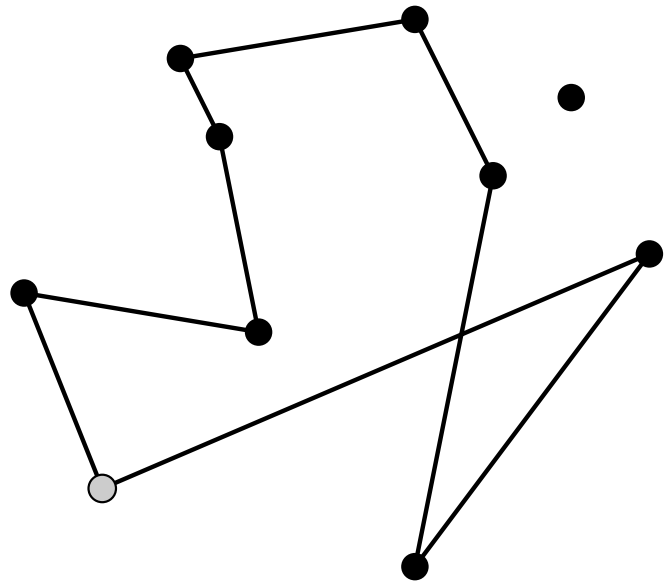
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



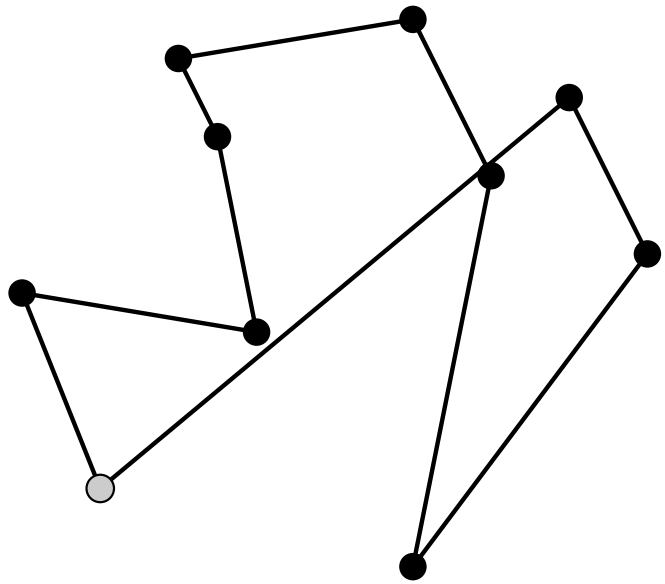
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



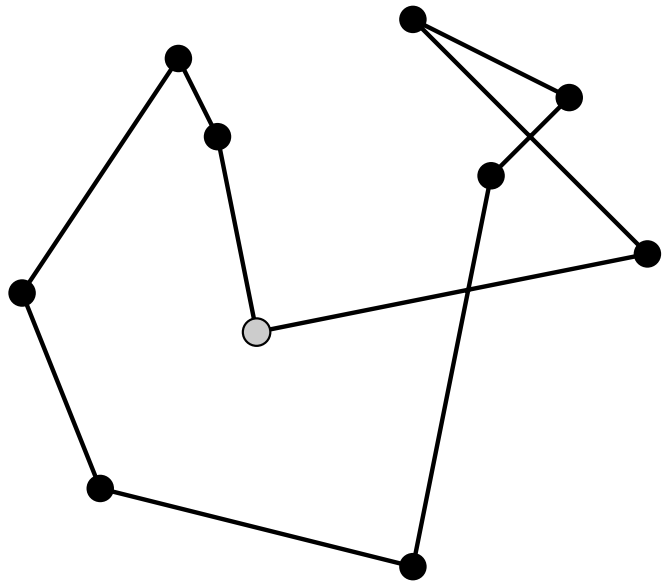
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



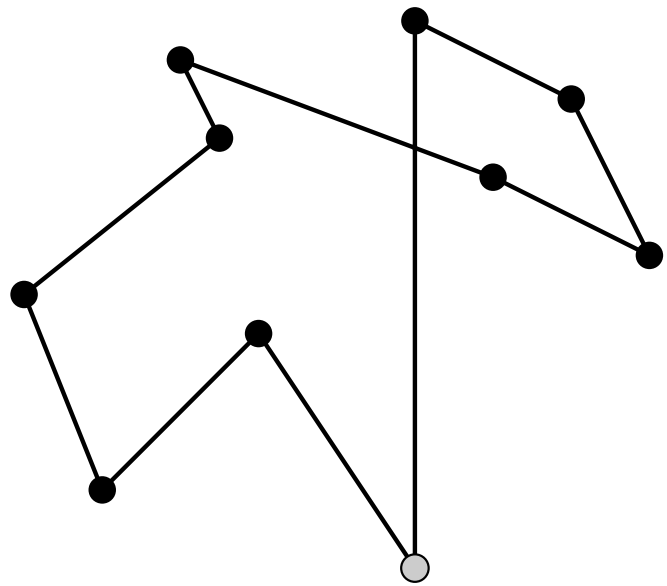
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



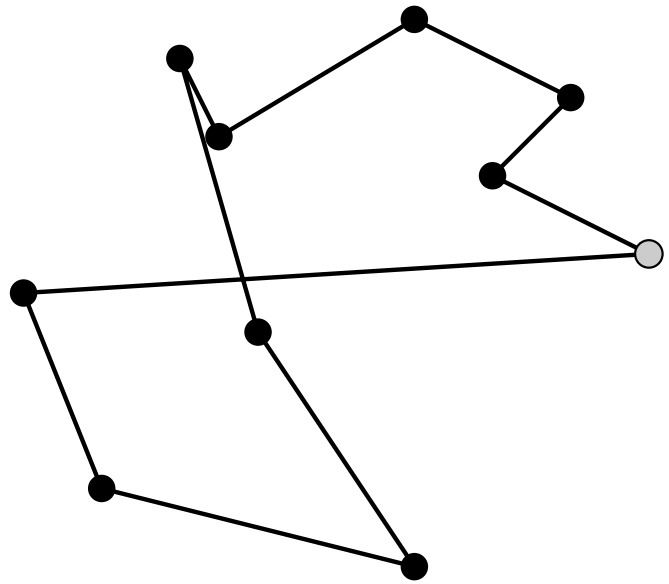
Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



Algorithme stochastique glouton

fonction GLOUTON(P)
- $S \leftarrow \emptyset$
- Construire la liste L des éléments insérables dans S à partir de P
- **tant que** $L \neq \emptyset$ **faire**
  - Évaluer le coût d'insertion de chaque élément de L
  - Retirer de L l'élément de coût minimal
  - Insérer cet élément dans S
- **retourner** S



Métaheuristique GRASP×ELS

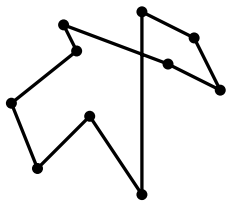
fonction GRASP(N, P)
    pour $i = 1..N$ **faire**
        $S_i \leftarrow$ HEURISTIQUECONSTRUCTIVE(P)
        $S_i \leftarrow$ RECHERCHELOCALE(P, S_i)
    $S^* \leftarrow$ SÉLECTION($\{S_1, S_2, \ldots, S_N\}$)
    retourner S^*

fonction ELS(P, S, N, M)
    $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
    $S^* \leftarrow S$
    pour $i = 1..N$ **faire**
        pour $j = 1..M$ **faire**
            $S_j \leftarrow$ MUTATION(S)
            $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
        $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
        $S^* \leftarrow$ SÉLECTION2(S^*, S')
        si CRITÈREACCEPTATION(S') **alors**
            $S \leftarrow S'$
    retourner S^*

Métaheuristique GRASP×ELS

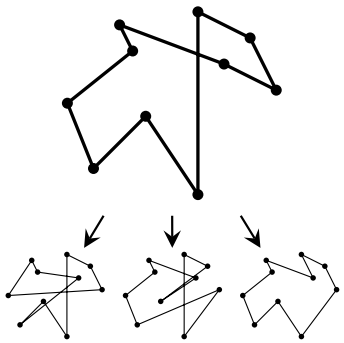
fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS



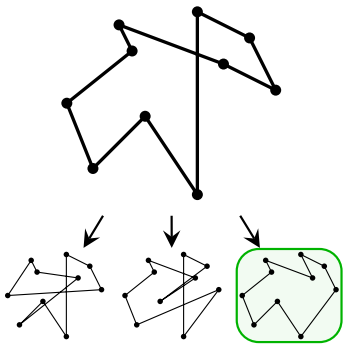
fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS



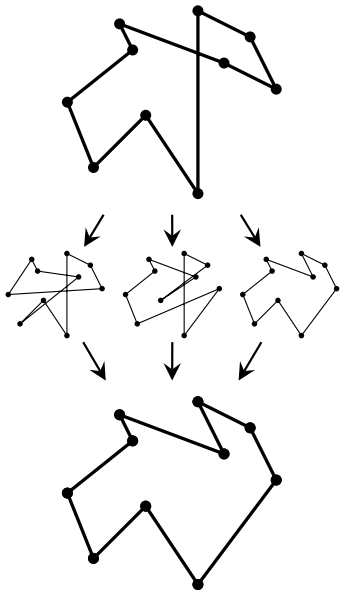
fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS



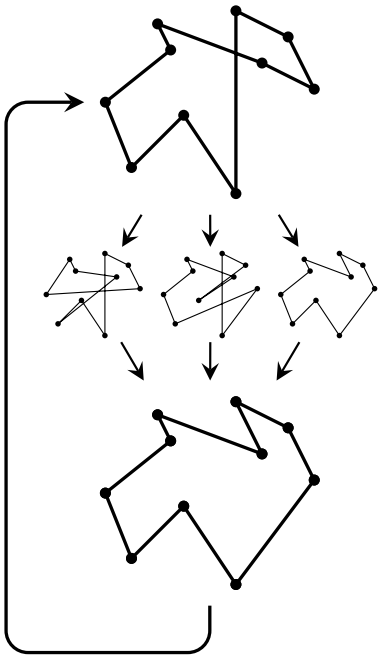
fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS



fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS



fonction ELS(P, S, N, M)
- $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
- $S^* \leftarrow S$
- **pour** $i = 1..N$ **faire**
  - **pour** $j = 1..M$ **faire**
    - $S_j \leftarrow$ MUTATION(S)
    - $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
  - $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
  - $S^* \leftarrow$ SÉLECTION2(S^*, S')
  - **si** CRITÈREACCEPTATION(S') **alors**
    - $S \leftarrow S'$
- **retourner** S^*

Métaheuristique GRASP×ELS

fonction GRASP×ELS(N, P)
    pour $i = 1..N$ **faire**
        $S_i \leftarrow$ HEURISTIQUECONSTRUCTIVE(P)
        $S_i \leftarrow$ ELS(P, S_i)
    $S^* \leftarrow$ SÉLECTION($\{S_1, S_2, \ldots, S_N\}$)
    retourner S^*

fonction ELS(P, S, N, M)
    $S \leftarrow$ RECHERCHELOCALEINITIALE(P, S)
    $S^* \leftarrow S$
    pour $i = 1..N$ **faire**
        pour $j = 1..M$ **faire**
            $S_j \leftarrow$ MUTATION(S)
            $S_j \leftarrow$ RECHERCHELOCALE(P, S_j)
        $S' \leftarrow$ SÉLECTION1($\{S_1, S_2, \ldots, S_M\}$)
        $S^* \leftarrow$ SÉLECTION2(S^*, S')
        si CRITÈREACCEPTATION(S') **alors**
            $S \leftarrow S'$
    retourner S^*

Points clés

Intérêt de cette application

- Plusieurs niveaux de parallélisation possible
- Stochastique $\Longrightarrow$ répétabilité difficile
- Composabilité (exemple : GRASP×ILS)

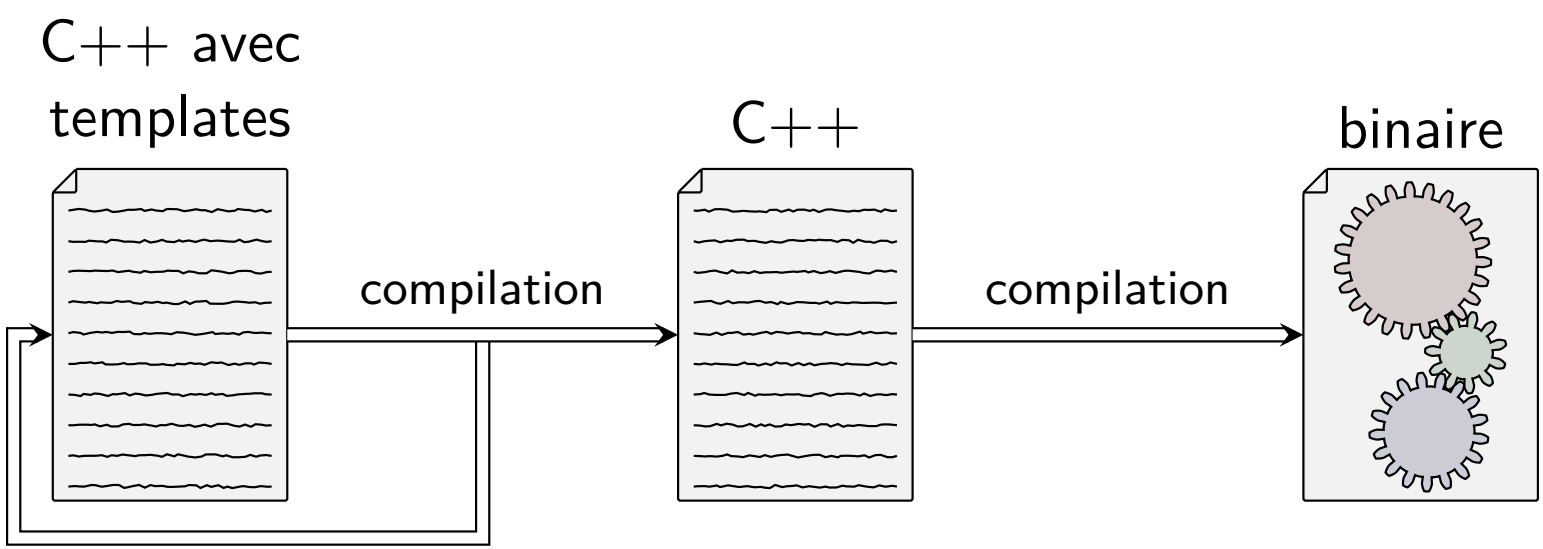
Métaprogrammation

Définition de métaprogramme

programme dont les données traitées sont des programmes

En C++ :

- notamment basé sur les templates
- exécution du métaprogramme durant la compilation
- génération d'un programme C++



Métaprogrammation template

Généricité

Résultats d'un template

- structure / classe / union
- fonction
- variable

Métaprogrammation template

Généricité

Résultats d'un template

- structure / classe / union
- fonction
- variable

```
template<typename T, std::size_t n>
struct Array {
  T data[n];
};
```

Métaprogrammation template

Généricité

Résultats d'un template

- structure / classe / union
- fonction
- variable

```
template<typename T, std::size_t n>
struct Array {
  T data[n];
};
```

Array<int, 5> Array<float, 3> Array<Foo, 256>

```
struct {
  int data[5];
};
```

```
struct {
  float data[3];
};
```

```
struct {
  Foo data[256];
};
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Assembleur produit

```
  lea  eax, [rsi-0x1]
  mov  r8d, 0x1
  test esi, esi
  jle  end
loop:
  imul r8d, edi
  sub  eax, 0x1
  jae  loop
end:
  mov  eax, r8d
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Assembleur produit

```
  lea  eax, [rsi-0x1]
  mov  r8d, 0x1
  test esi, esi
  jle  end
loop:
r *= x ⟶ imul r8d, edi
  sub  eax, 0x1
  jae  loop
end:
  mov  eax, r8d
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Assembleur produit

```
  lea  eax, [rsi-0x1]
  mov  r8d, 0x1
  test esi, esi
  jle  end
loop:
  imul r8d, edi
  sub  eax, 0x1
  jae  loop
end:
  mov  eax, r8d
```

n-- → sub eax, 0x1

Puissance avec exposant connu

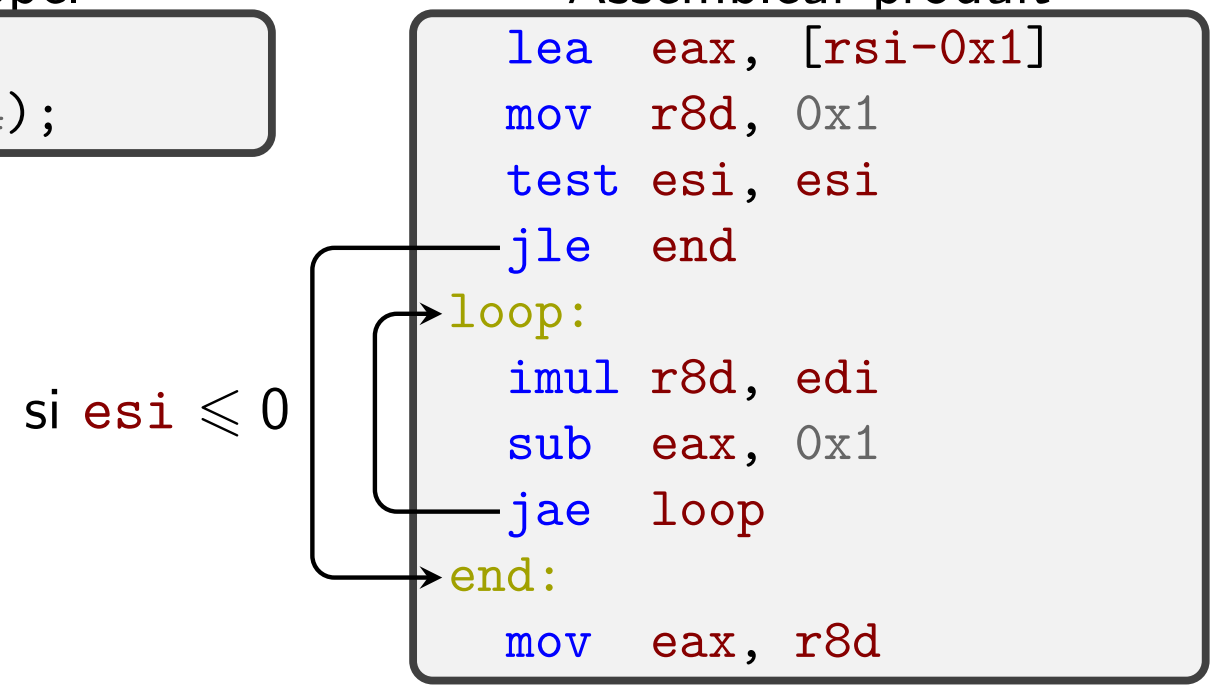
Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Assembleur produit



Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

Assembleur produit

```
  lea  eax, [rsi-0x1]
  mov  r8d, 0x1
  test esi, esi
  jle  end
loop:
  imul r8d, edi
  sub  eax, 0x1
  jae  loop
end:
  mov  eax, r8d
```

tant que eax > 0

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Exemple d'appel

```
int x = 2;
int r = pow(x, 24);
```

- *n* multiplications
- *n* soustractions
- structures de contrôle

Assembleur produit

```
  lea  eax, [rsi-0x1]
  mov  r8d, 0x1
  test esi, esi
  jle  end
loop:
  imul r8d, edi
  sub  eax, 0x1
  jae  loop
end:
  mov  eax, r8d
```

Puissance avec exposant connu

Fonction puissance

```
int pow(int x, int n) {
  int r = 1;
  while(n-- > 0) r *= x;
  return r;
}
```

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

C++ généré

```
pow<24>(x)
```

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

C++ généré

```
x* pow<23>(x)
```

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

C++ généré

```
x**x* pow<22>(x)
```

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

C++ généré

```
x*...*x* pow<0>(x)
```

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

C++ généré

$$\underbrace{x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x}_{\text{24 occurrences de x}}*1$$

Puissance avec exposant connu

Métafonction puissance

```
template<unsigned int n>
constexpr int pow(int x) {
  return x * pow<n-1>(x);
}

template<>
constexpr int pow<0>(int) {
  return 1;
}
```

Exemple d'appel

```
int x = 2;
int r = pow<24>(x);
```

Assembleur produit

```
imul ecx, ecx ; x^2
imul ecx, eax ; x^3
imul ecx, ecx ; x^6
imul ecx, ecx ; x^12
imul ecx, ecx ; x^24
```

$\Longrightarrow$ exponentiation rapide par le compilateur

Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

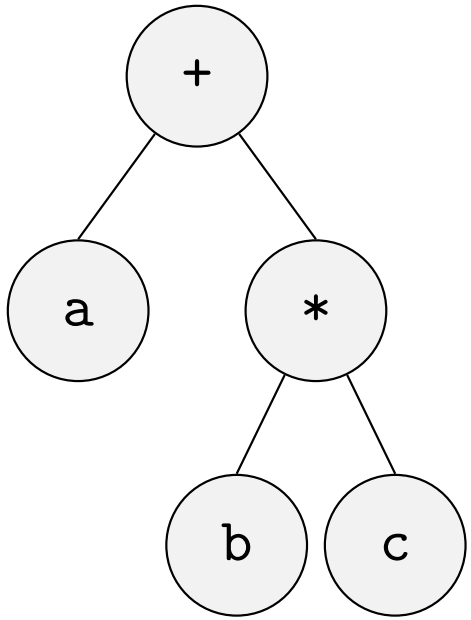
```
auto e = a + b * c;
```

Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

```
auto e = a + b * c;
```



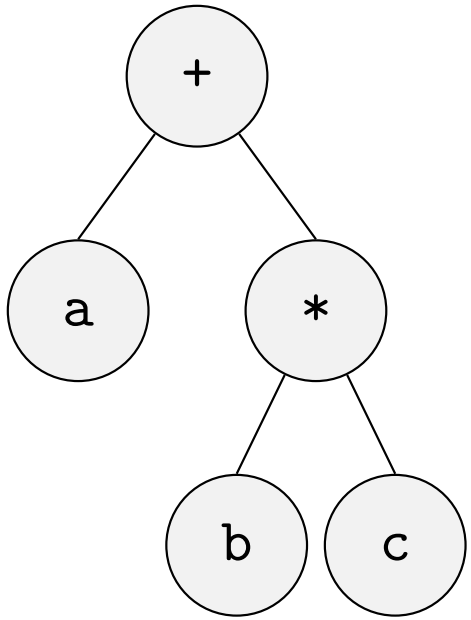
Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

```
auto e = a + b * c;
```

```
Expr<op::Add,
  A,
  Expr<op::Mul,
    B,
    C
  >
>
```



Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

```
auto e = a + b * c;
```

```
Expr<op::Add,
  A,
  Expr<op::Mul,
    B,
    C
  >
>
```

```
template<typename Lhs, typename Rhs>
auto operator*(Lhs, Rhs) {
  return Expr<op::Mul, Lhs, Rhs>{};
}
```

```
template<typename Lhs, typename Rhs>
auto operator+(Lhs, Rhs) {
  return Expr<op::Add, Lhs, Rhs>{};
}
```

Patrons d'expression

Concept

- traitement d'arbre syntaxiques abstraits
- acquisition avec un langage embarqué

Applications

- Eigen : optimisation de calculs matriciels
- Adept : optimisation de dérivées de fonctions
- Parallélisation assistée/automatique

Patrons d'expression

Exemple d'optimisation

```
5 * (2 * x + 1) - (x + 3)
```

Patrons d'expression

Exemple d'optimisation

```
5 * (2 * x + 1) - (x + 3)
```

```
Expr<op::Sub,
  Expr<op::Mul,
    Cst<5>,
    Expr<op::Add,
      Expr<op::Mul,
        Cst<2>, x
      >
      Cst<1>
    >
  >,
  Expr<op::Add,
    x, Cst<3>
  >
>
```

Patrons d'expression

Exemple d'optimisation

```
5 * (2 * x + 1) - (x + 3)
```

```
Expr<op::Sub,
  Expr<op::Mul,
    Cst<5>,
    Expr<op::Add,
      Expr<op::Mul,
        Cst<2>, x
      >
      Cst<1>
    >
  >,
  Expr<op::Add,
    x, Cst<3>
  >
>
```

Motif « affine »

```
Lin<a, b> ->
Expr<op::Add,
  Expr<op::Mul, Cst<a>, x>,
  Cst<b>
>
```

Motif « constante × affine »

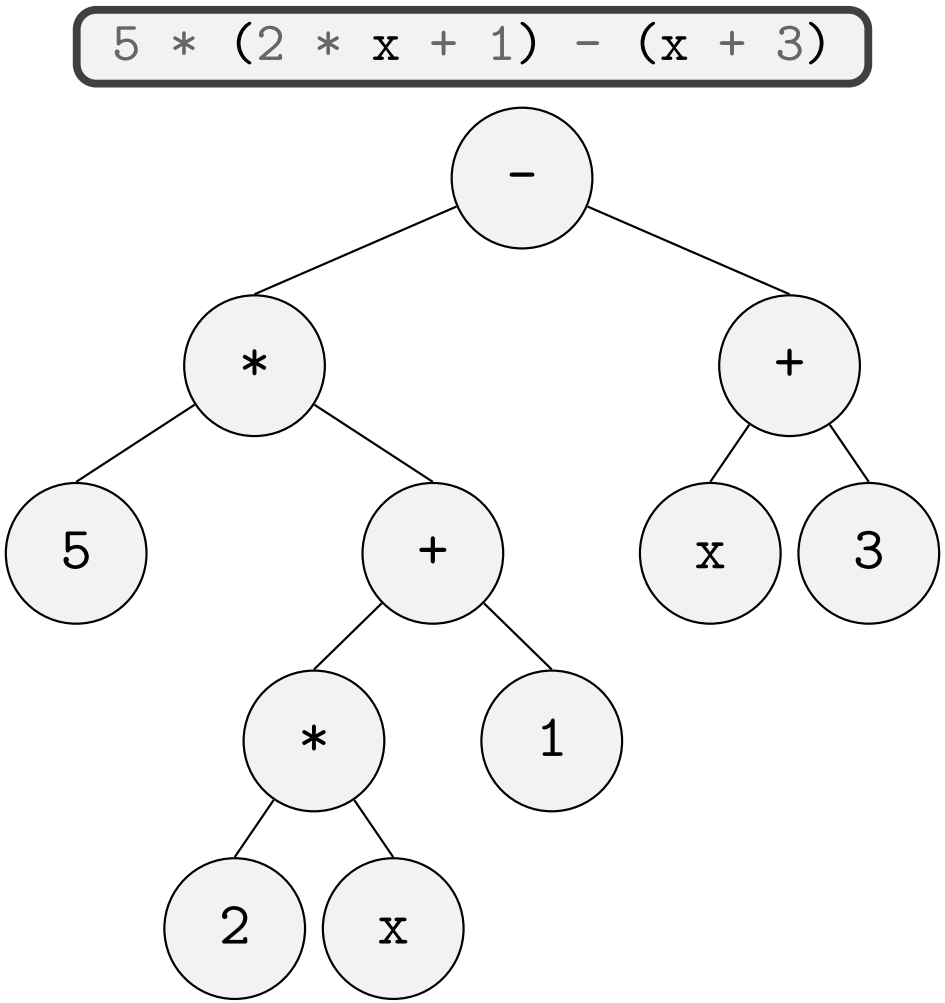
```
Opt<op::Mul, Cst<c>, Lin<a, b>>
-> Lin<a*c, b*c>
```

Motif « affine − affine »

```
Opt<op::Sub, Lin<a, b>, Lin<c, d>>
-> Lin<a-c, b-d>
```

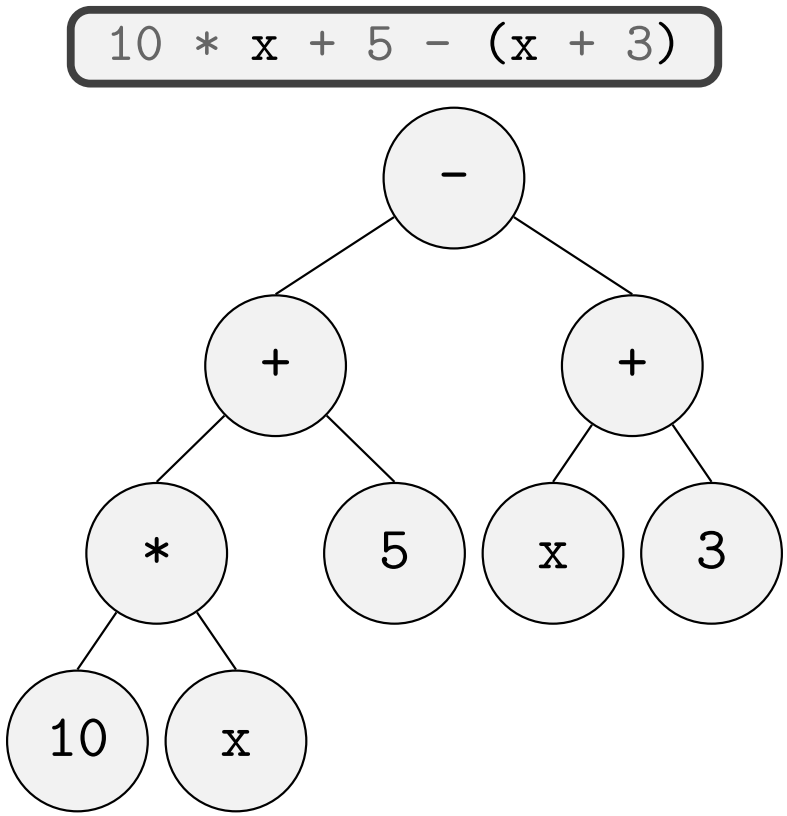
Patrons d'expression

Exemple d'optimisation



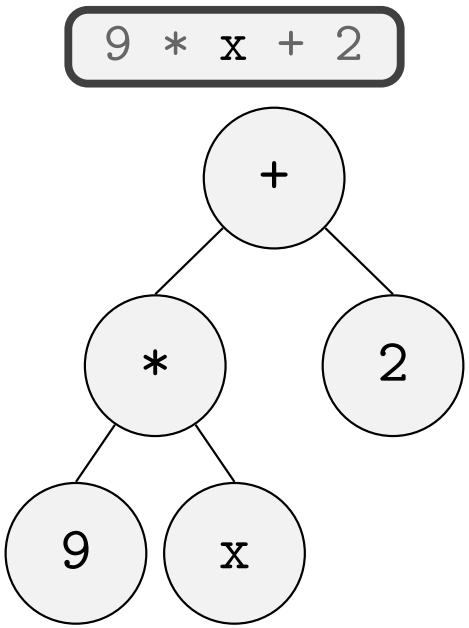
Patrons d'expression

Exemple d'optimisation



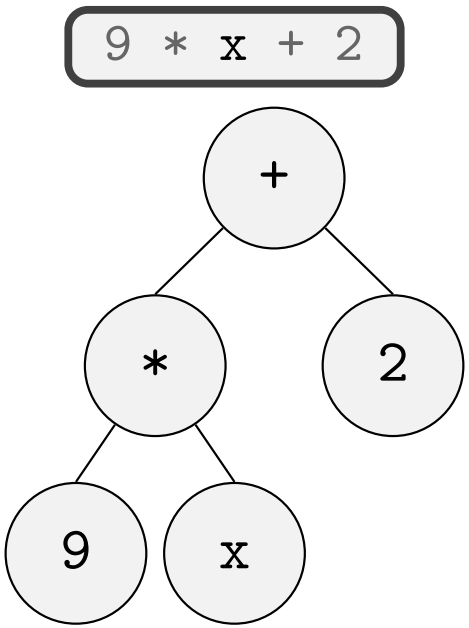
Patrons d'expression

Exemple d'optimisation



Patrons d'expression

Exemple d'optimisation



- code effectif produit
- analyse durant la compilation

Atouts de la métaprogrammation template

- Analyse et génération de code durant la compilation
- Possibilité de le faire sans surcoût
- C++ standard

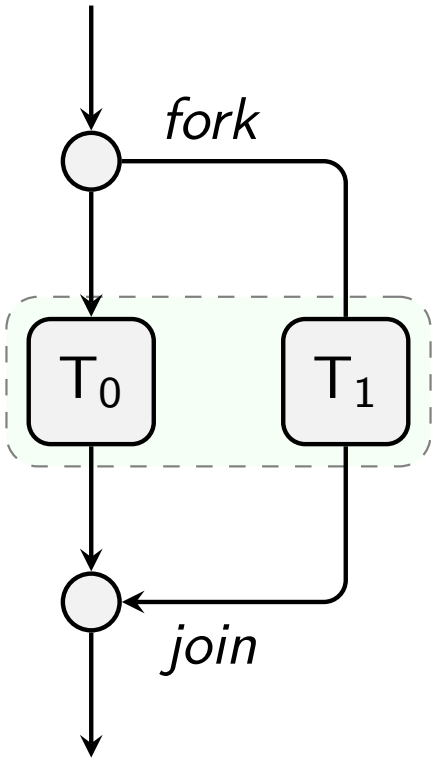
Plan

Concept

- Motifs d'exécution parallèle
- Composition

Concept

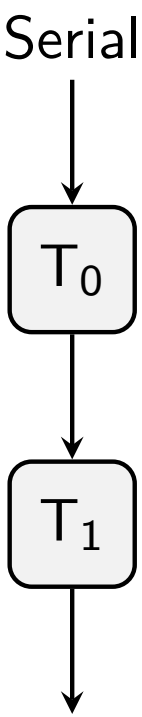
- Motifs d'exécution parallèle
- Composition

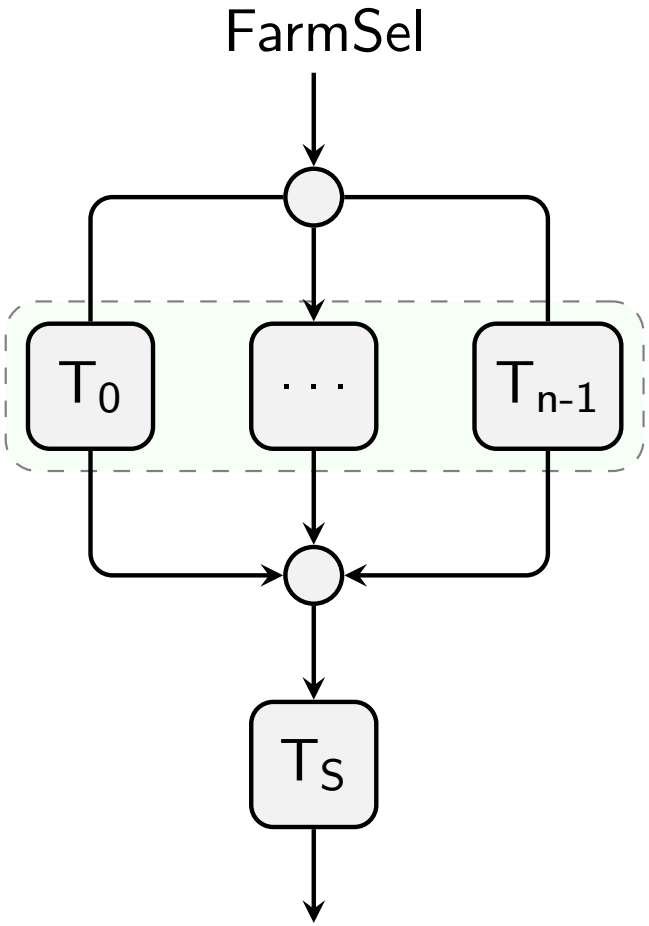


Concept

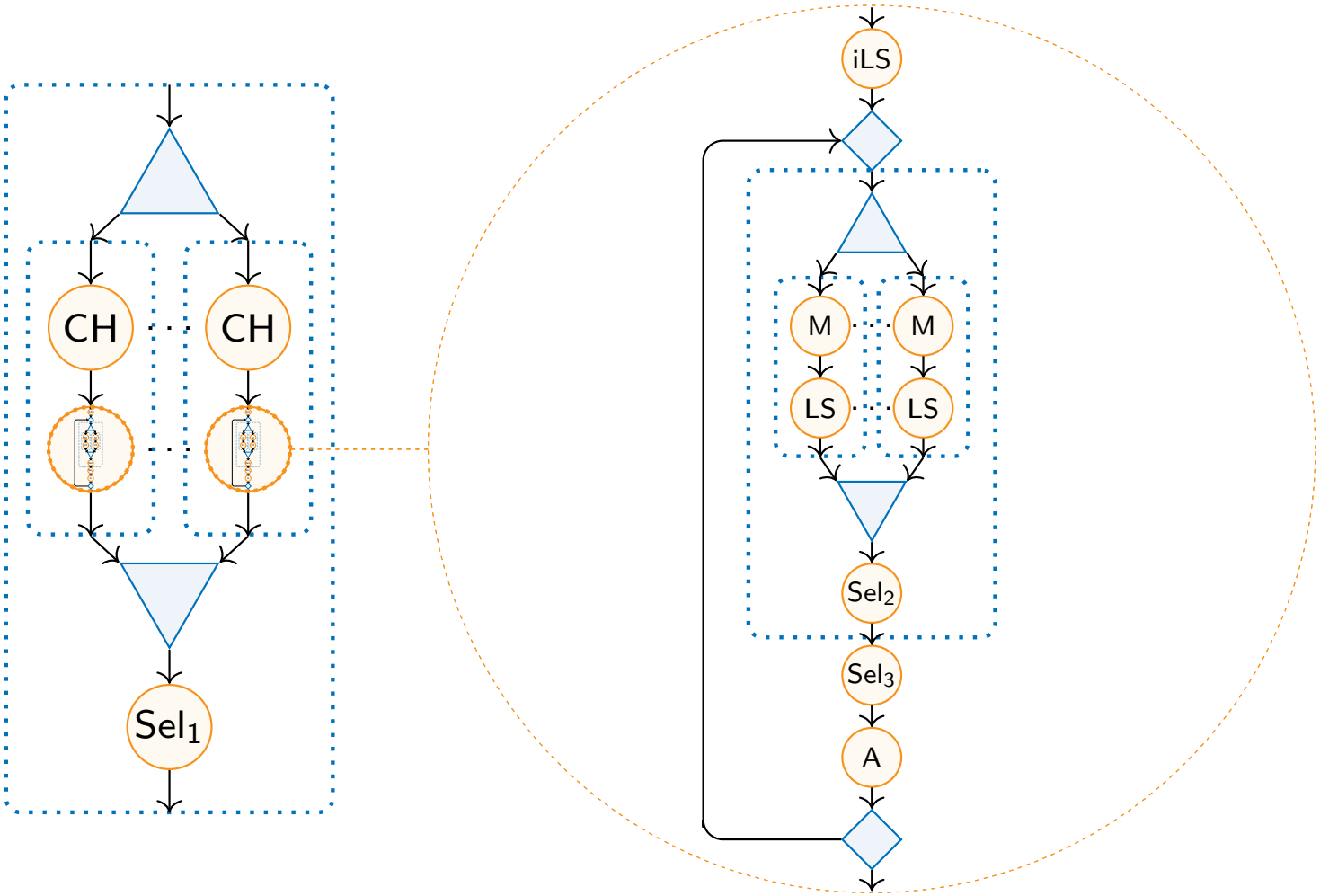
- Motifs d'exécution parallèle
- Composition

→ GRASP×ELS

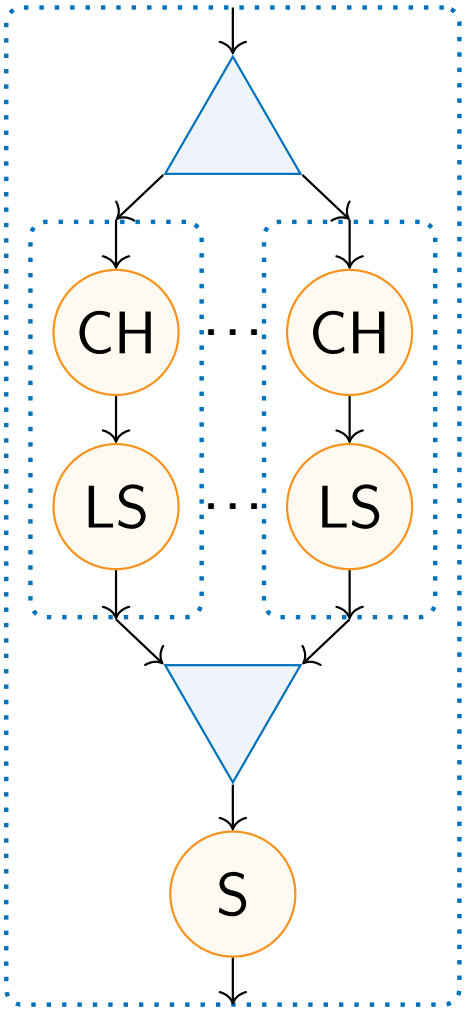




GRASP×ELS



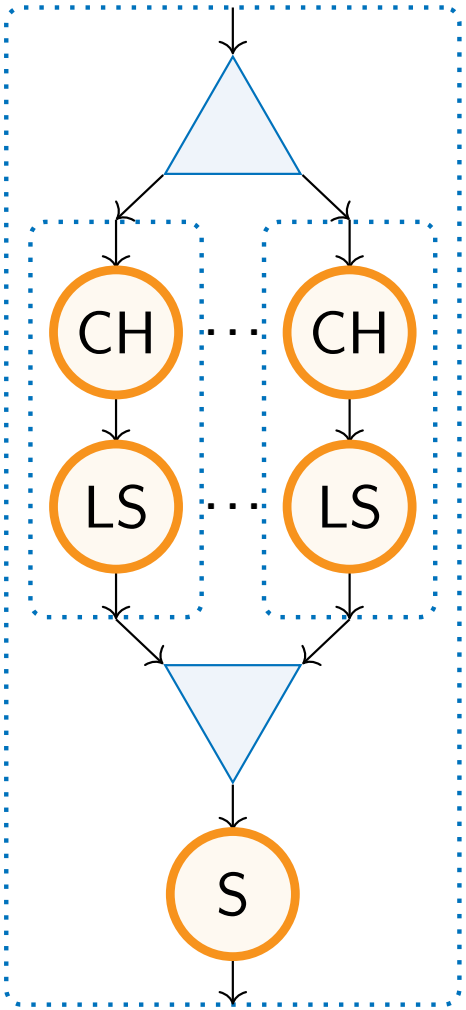
Composition



- Muscles
- Structure
- Liens

fonction $\mathrm{GRASP}(N, P)$
  pour $i = 1..N$ **faire**
    $S_i \leftarrow \textsc{heuristiqueConstructive}(P)$
    $S_i \leftarrow \textsc{rechercheLocale}(P, S_i)$
  $S^* \leftarrow \textsc{sélection}(\{S_1, S_2, \ldots, S_N\})$
  retourner S^*

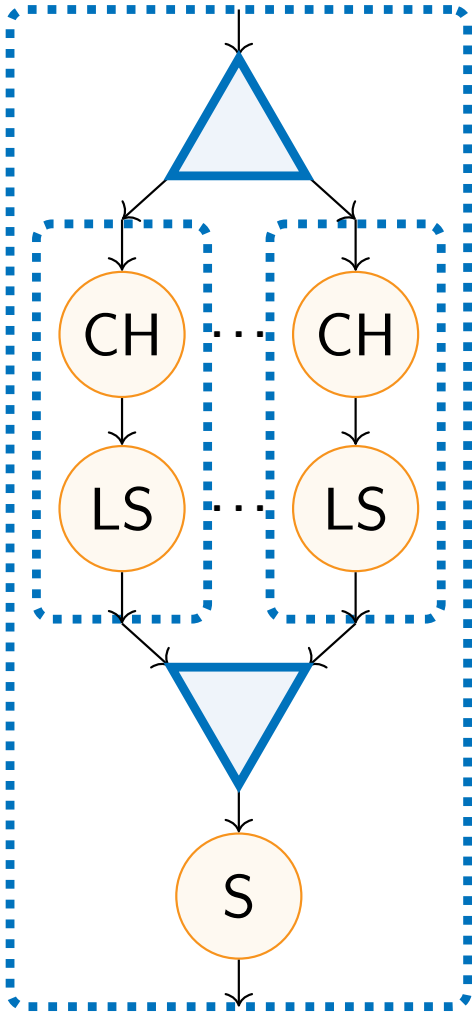
Composition



- Muscles : fonctions « domaine »
- Structure
- Liens

fonction GRASP(N, P)
  pour $i = 1..N$ **faire**
    $S_i \leftarrow$ HEURISTIQUECONSTRUCTIVE(P)
    $S_i \leftarrow$ RECHERCHELOCALE(P, S_i)
  $S^* \leftarrow$ SÉLECTION$(\{S_1, S_2, \ldots, S_N\})$
  retourner S^*

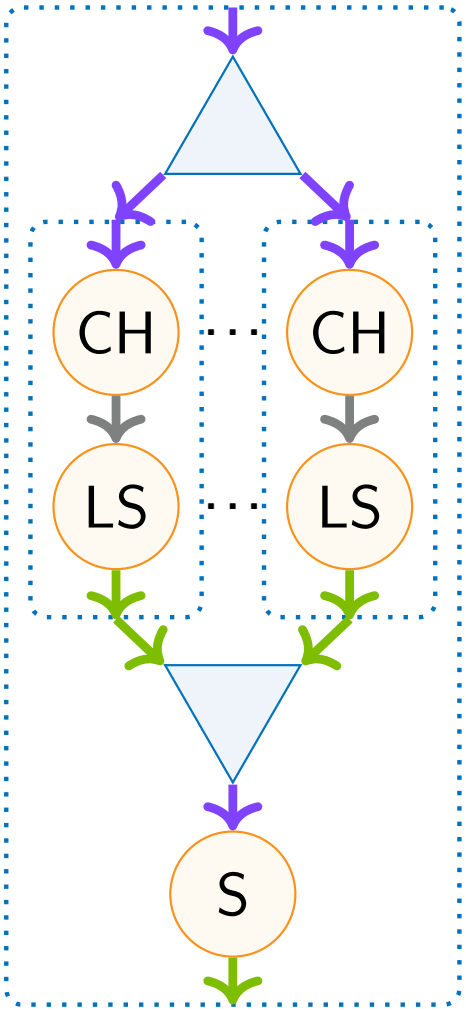
Composition



- Muscles : fonctions « domaine »
- Structure : motifs d'exécution
- Liens

fonction GRASP(N, P)
  pour $i = 1..N$ **faire**
    $S_i \leftarrow$ HEURISTIQUECONSTRUCTIVE(P)
    $S_i \leftarrow$ RECHERCHELOCALE(P, S_i)
  $S^* \leftarrow$ SÉLECTION($\{S_1, S_2, \ldots, S_N\}$)
  retourner S^*

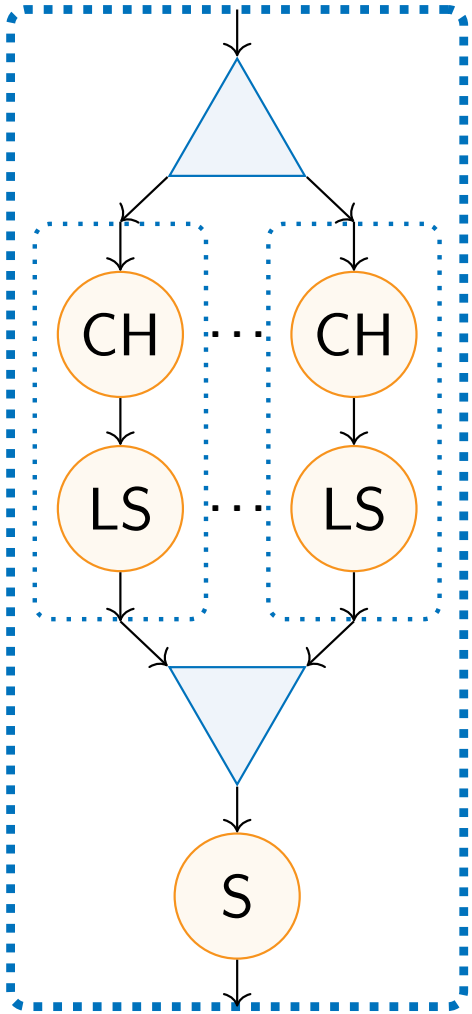
Composition



- Muscles : fonctions « domaine »
- Structure : motifs d'exécution
- Liens : flux de données

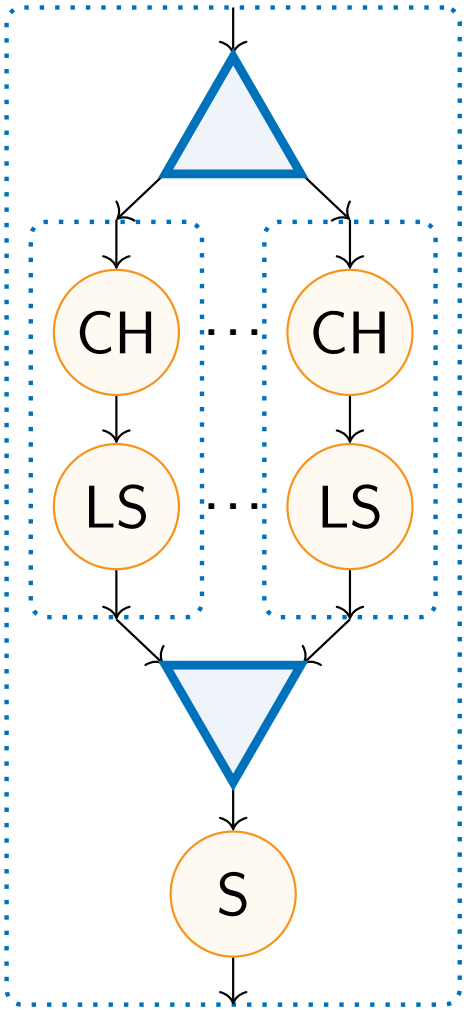
fonction GRASP(N, P)
    pour $i = 1..N$ **faire**
        $S_i \leftarrow$ HEURISTIQUECONSTRUCTIVE(P)
        $S_i \leftarrow$ RECHERCHELOCALE(P, S_i)
    $S^* \leftarrow$ SÉLECTION($\{S_1, S_2, \ldots, S_N\}$)
    retourner S^*

Structure



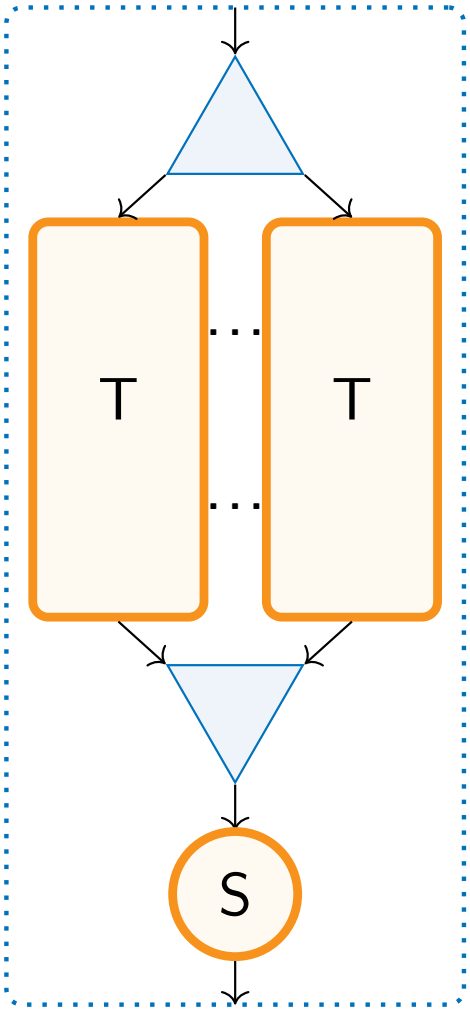

```
template<
  typename CH, typename LS, typename S
>
using SkelGraspStructure =
S<FarmSel,
  S<Serial,
    CH, LS
  >,
  S
>;
```

Structure



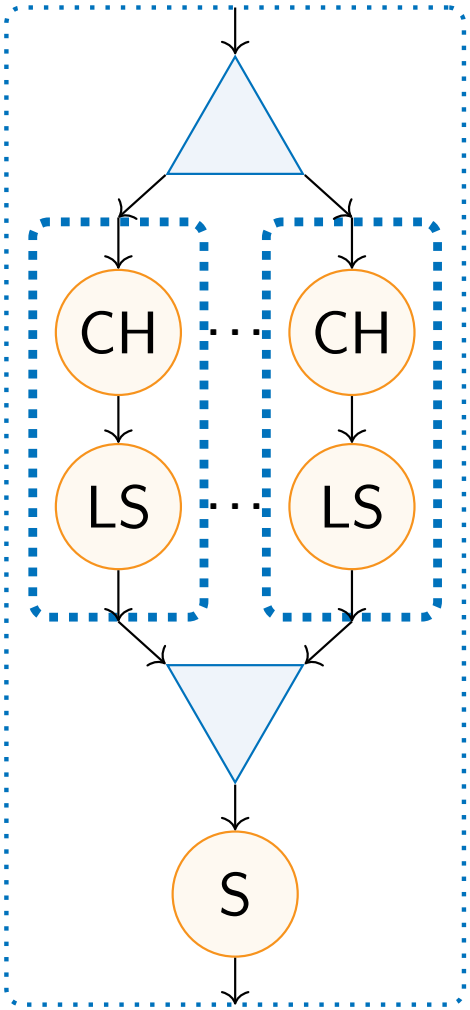

```
template<
  typename CH, typename LS, typename S
>
using SkelGraspStructure =
S<FarmSel,
  S<Serial,
    CH, LS
  >,
  S
>;
```

Structure



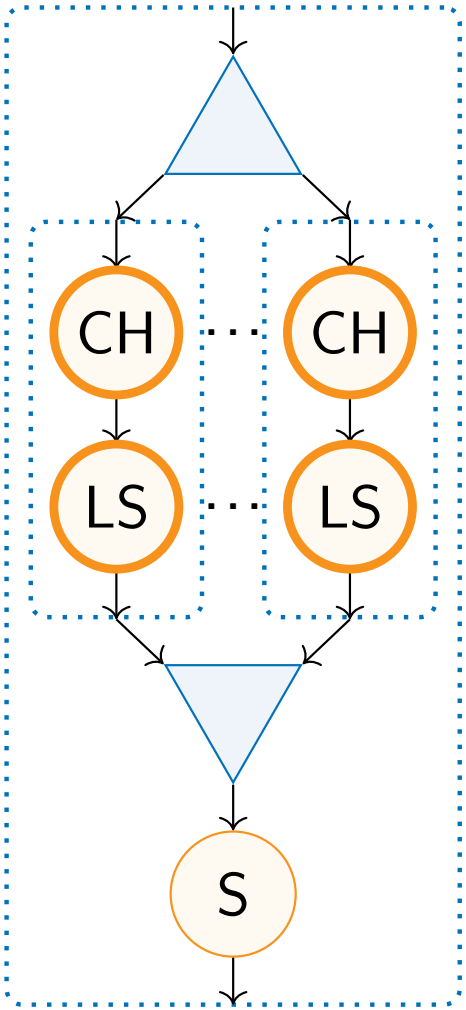

```
template<
  typename CH, typename LS, typename S
>
using SkelGraspStructure =
S<FarmSel,
  S<Serial,
    CH, LS
  >,
  S
>;
```

Structure



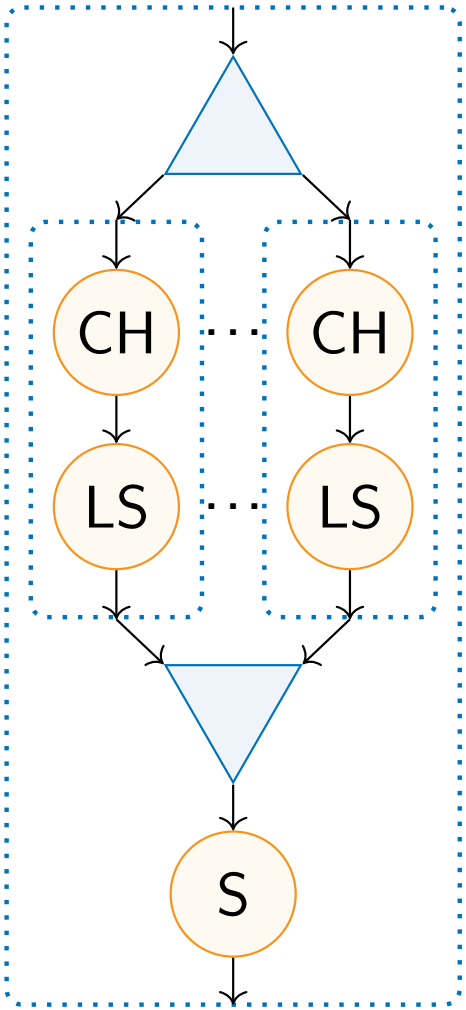

```
template<
  typename CH, typename LS, typename S
>
using SkelGraspStructure =
S<FarmSel,
  S<Serial,
    CH, LS
  >,
  S
>;
```

Structure




```
template<
  typename CH, typename LS, typename S
>
using SkelGraspStructure =
S<FarmSel,
  S<Serial,
    CH, LS
  >,
  S
>;
```

Liens



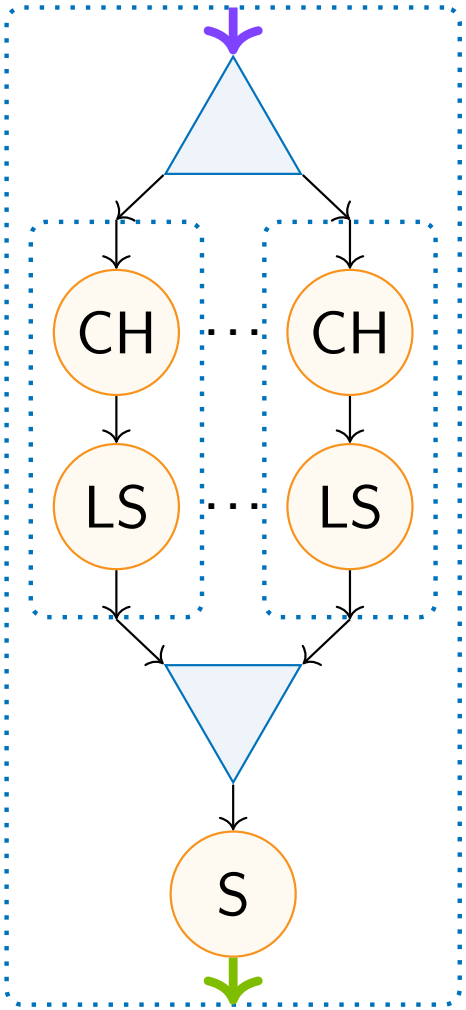
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



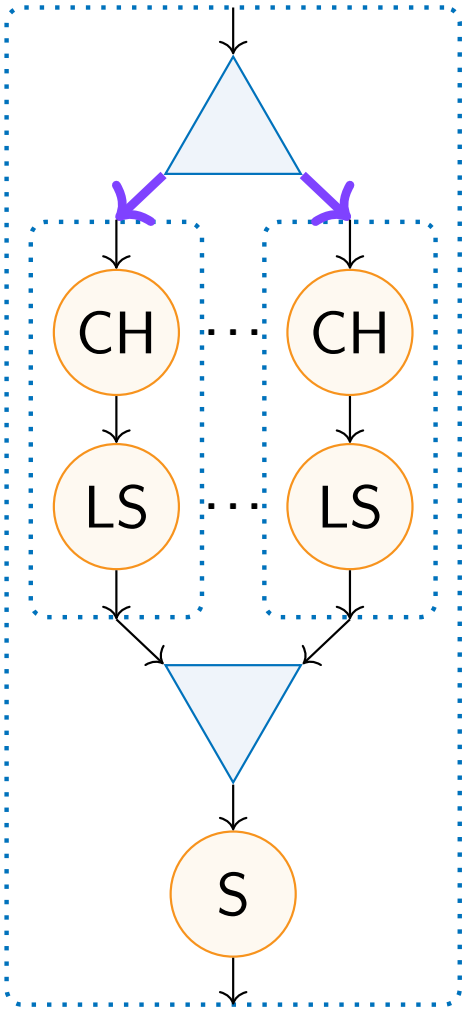
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



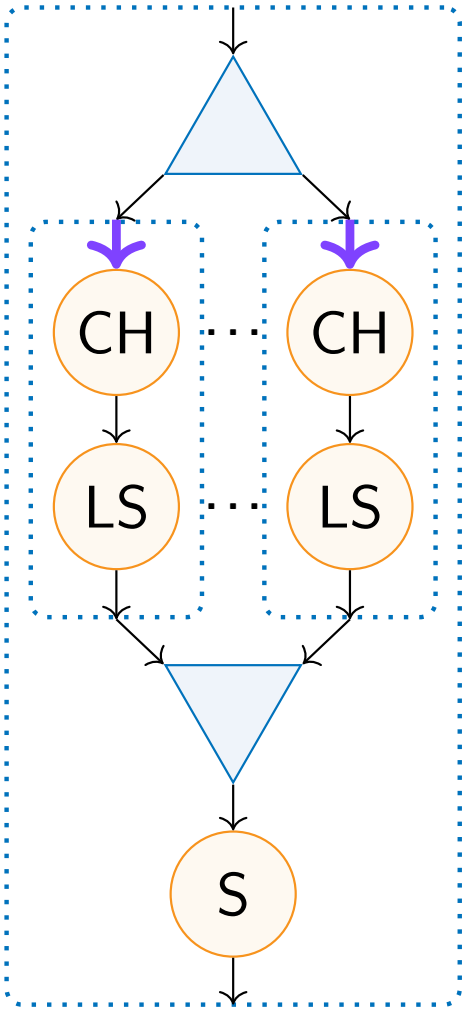
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



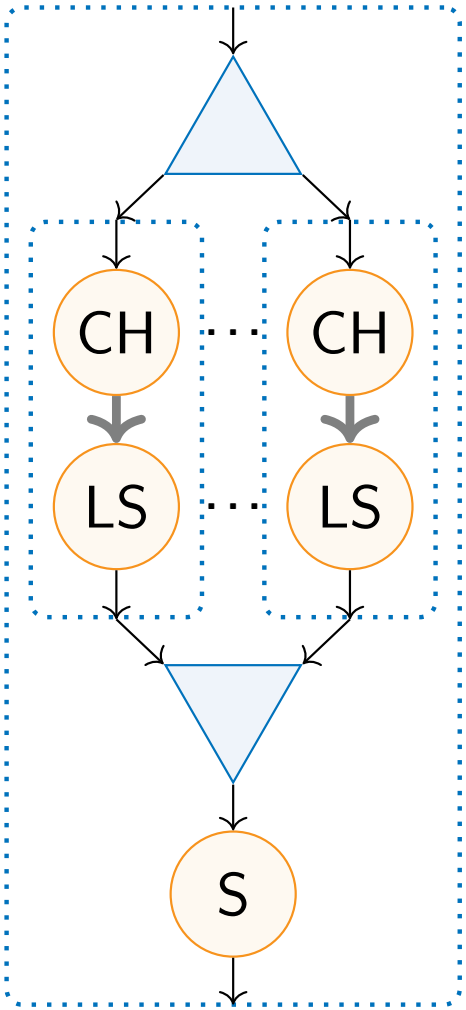
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



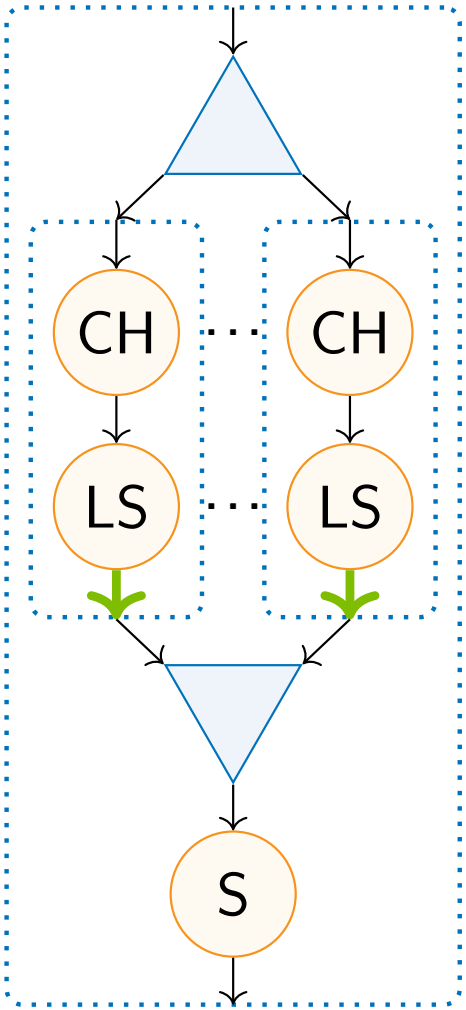
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



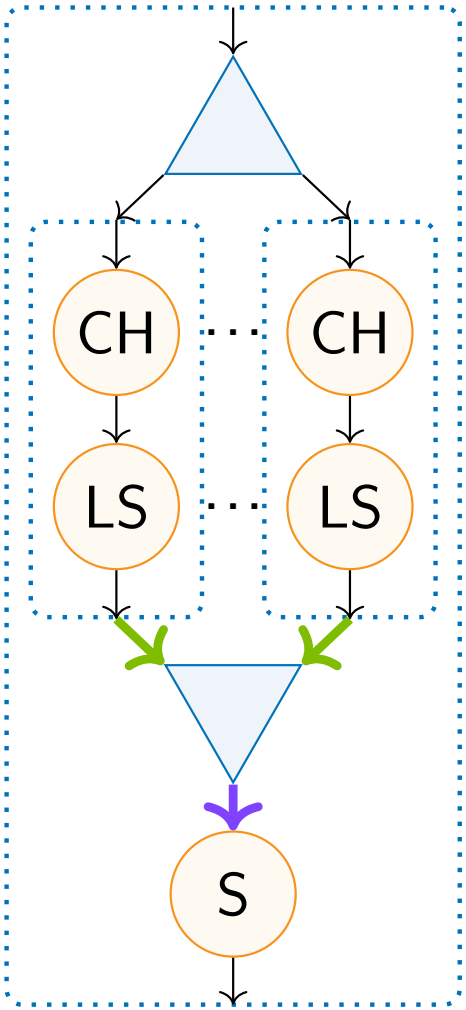
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



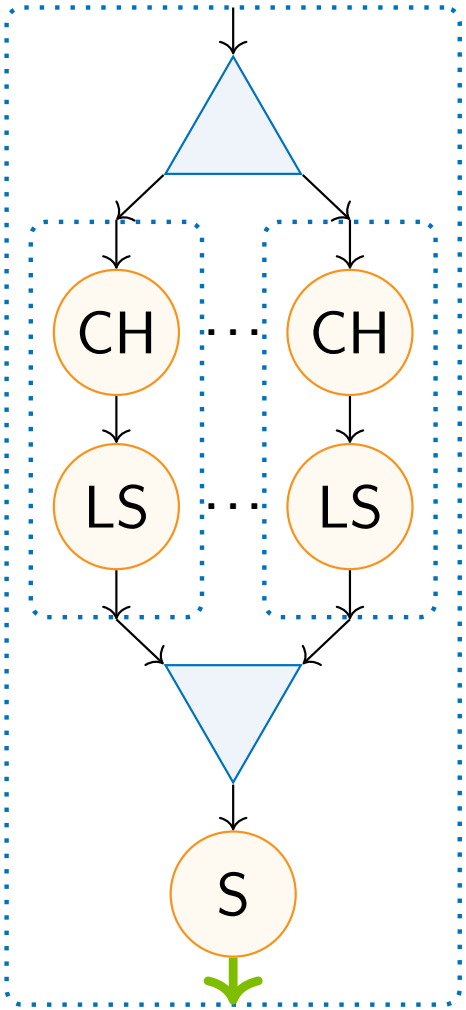
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



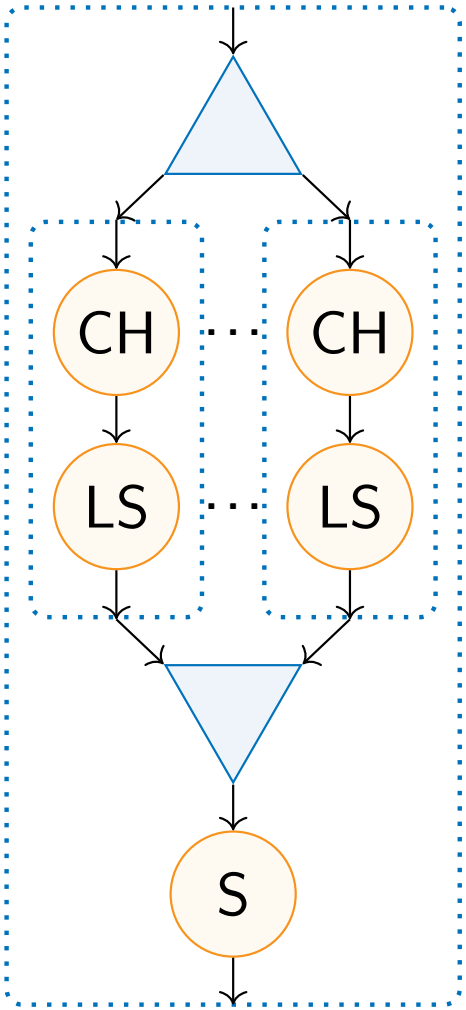
`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Liens



`P<i>` paramètre d'indice `i`

`R<i>` résultat d'indice `i`

`RNG` générateur de nombres pseudo-aléatoires

```
template<
  typename Problem, typename Solution
>
using SkelGraspLinks =
L<FarmSel, Solution(Problem),
  L<Serial, R<1>(P<0>),
    Solution(P<0>, RNG),
    Solution(R<0>, RNG)
  >,
  Solution(Solution, Solution)
>;
```

Corps : instanciation d'un squelette

$\texttt{SkelEls} = \texttt{SkelElsStruct} + \texttt{SkelElsLinks}$

```
using Els = SkelEls<
  tsp::Problem, tsp::Solution,
  iLS, M, LS, sel2, sel3, A
>;
```

Corps : instanciation d'un squelette

`SkelEls` = `SkelElsStruct` + `SkelElsLinks`

```
using Els = SkelEls<
  tsp::Problem, tsp::Solution,
  iLS, M, LS, sel2, sel3, A
>;
```

Corps : instanciation d'un squelette

$$\texttt{SkelEls} = \texttt{SkelElsStruct} + \texttt{SkelElsLinks}$$

```
using Els = SkelEls<
  tsp::Problem, tsp::Solution,
  iLS, M, LS, sel2, sel3, A
>;
```

Corps : instanciation d'un squelette

SkelEls = SkelElsStruct + SkelElsLinks

```
using Els = SkelEls<
  tsp::Problem, tsp::Solution,
  iLS, M, LS, sel2, sel3, A
>;
```

```
using GraspEls = SkelGrasp<
  tsp::Problem, tsp::Solution,
  Greedy, Els, sel1
>;
```

Exécution

```
auto graspEls = implement<DynamicPool, GraspEls>();
```

Exécution

```
auto graspEls = implement<DynamicPool, GraspEls>();

graspEls.skeleton.n = 10;
graspEls.skeleton.task.task<1>().task<1>().n = 15;
graspEls.skeleton.task.task<1>().task<1>().task.n = 20;
```

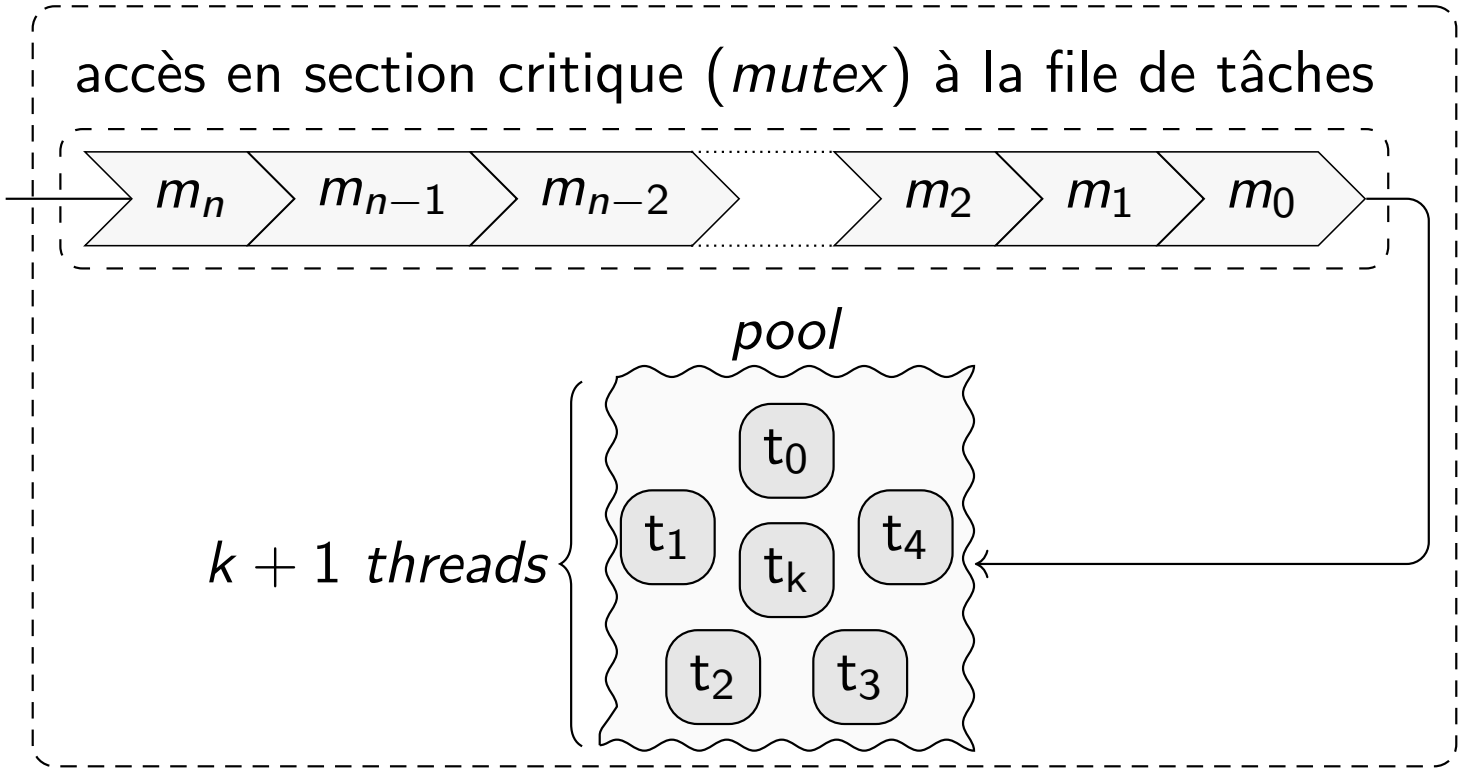
Exécution

```
auto graspEls = implement<DynamicPool, GraspEls>();

graspEls.skeleton.n = 10;
graspEls.skeleton.task.task<1>().task<1>().n = 15;
graspEls.skeleton.task.task<1>().task<1>().task.n = 20;

graspEls.executor.cores = 16;
```

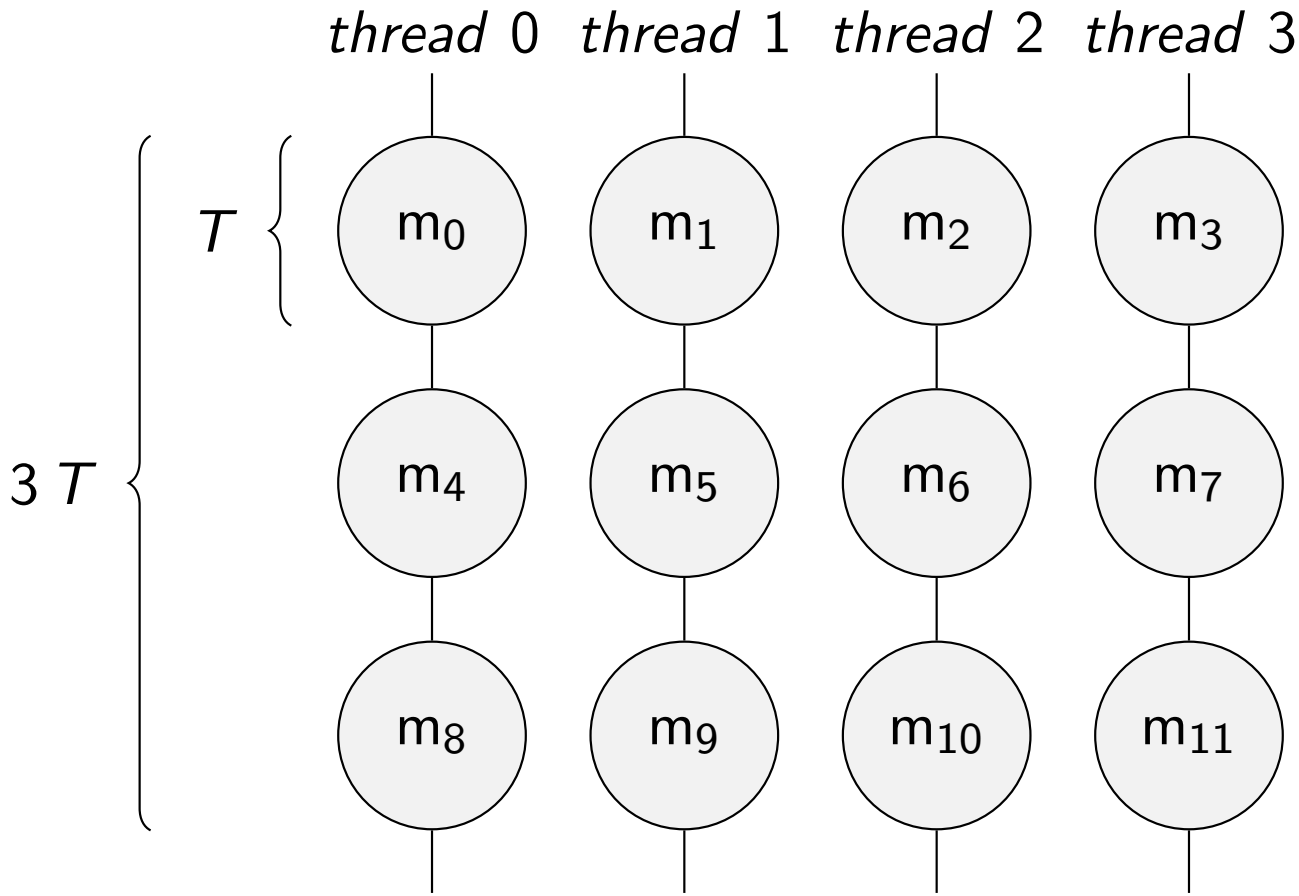
Exécution

```
auto graspEls = implement<DynamicPool, GraspEls>();

graspEls.skeleton.n = 10;
graspEls.skeleton.task.task<1>().task<1>().n = 15;
graspEls.skeleton.task.task<1>().task<1>().task.n = 20;

graspEls.executor.cores = 16;

tsp::Solution solution = graspEls(problem);
```

`DynamicPool` : *Thread pool*



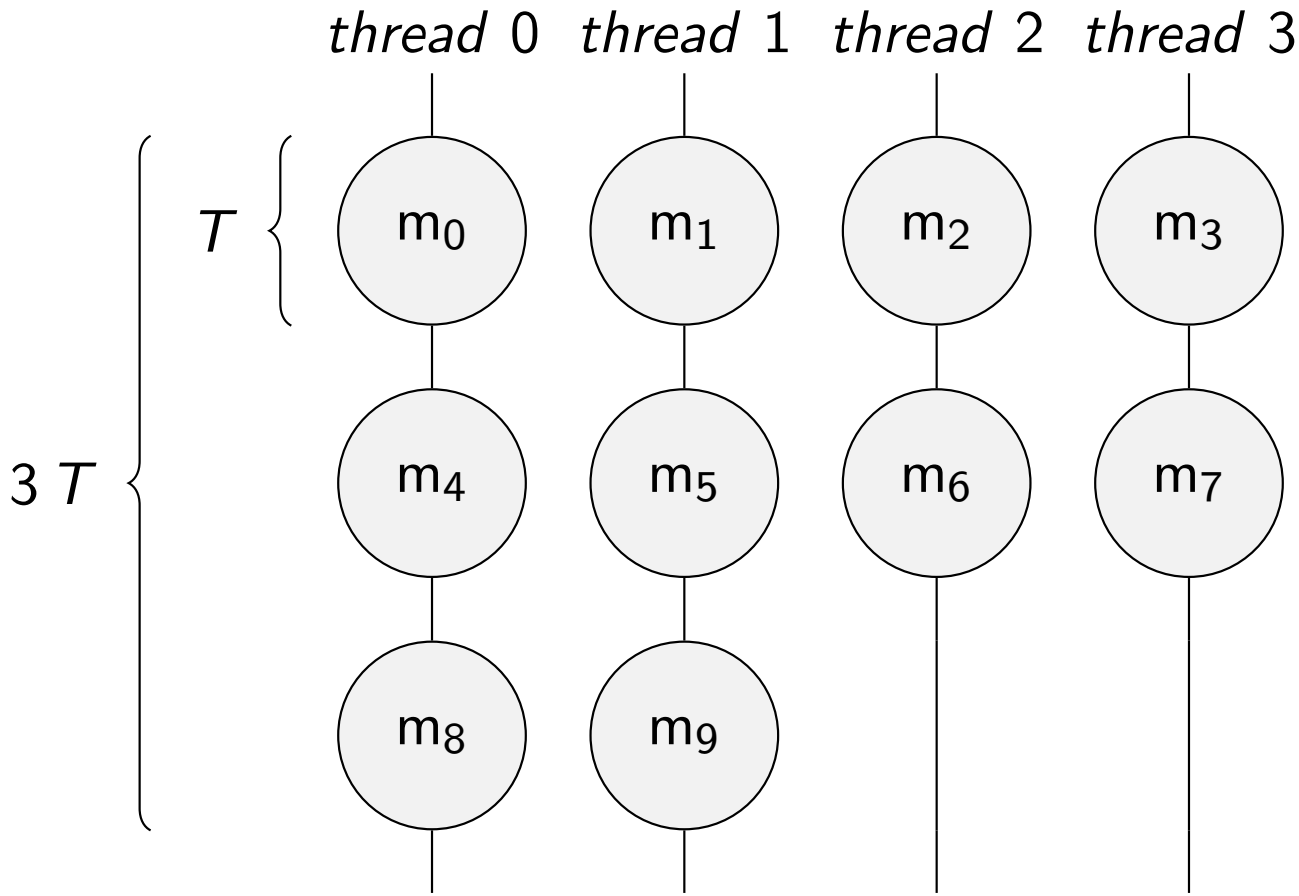
Répartition équilibrée



12 tâches = 3 tâches/*thread* × 4 *threads* + 0 tâche

$\Longrightarrow$ accélération idéale : 4

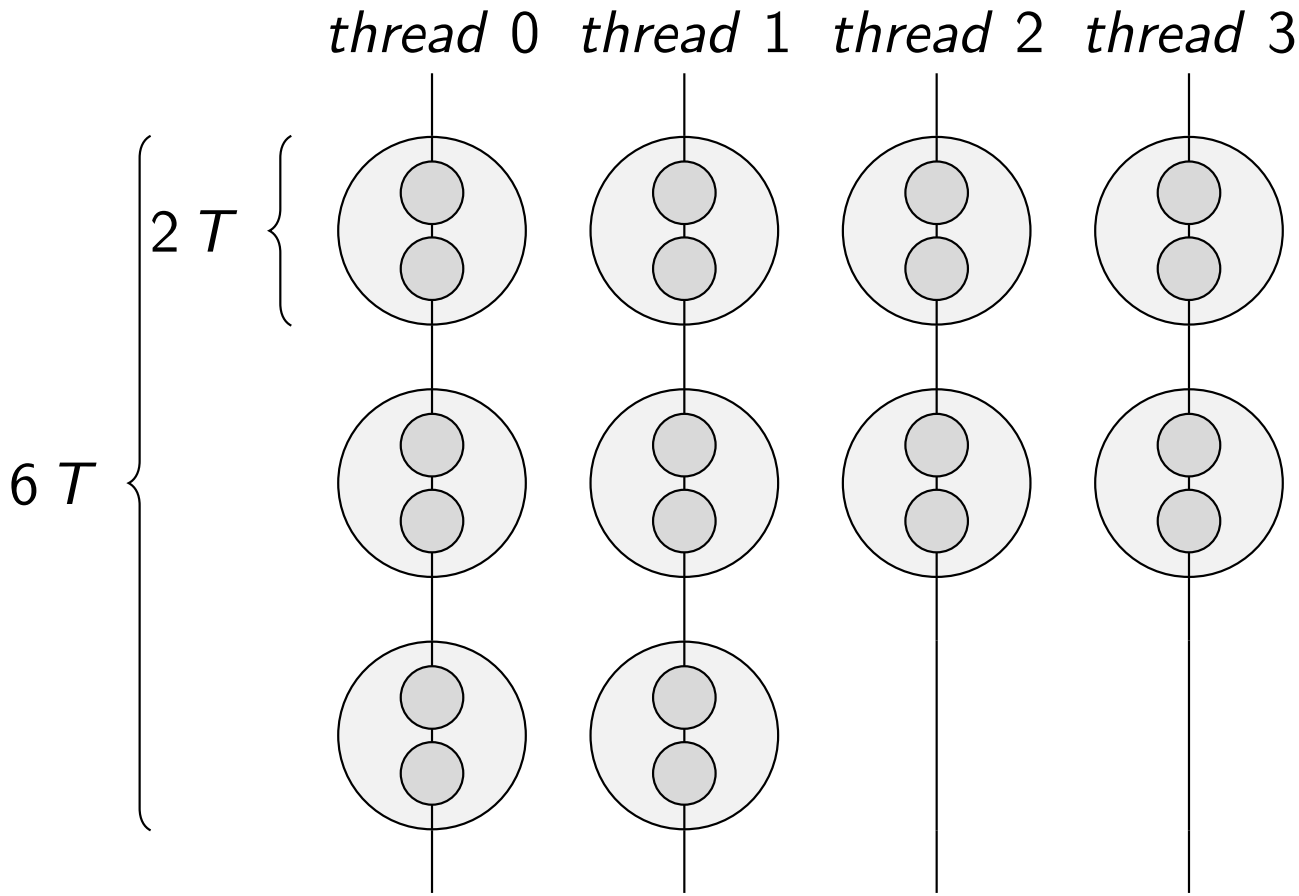
Répartition équilibrée



10 tâches $= 2$ tâches$/$*thread* $\times 4$ *threads* $+ 2$ tâches

$\Longrightarrow$ accélération idéale : $3.\overline{3}$

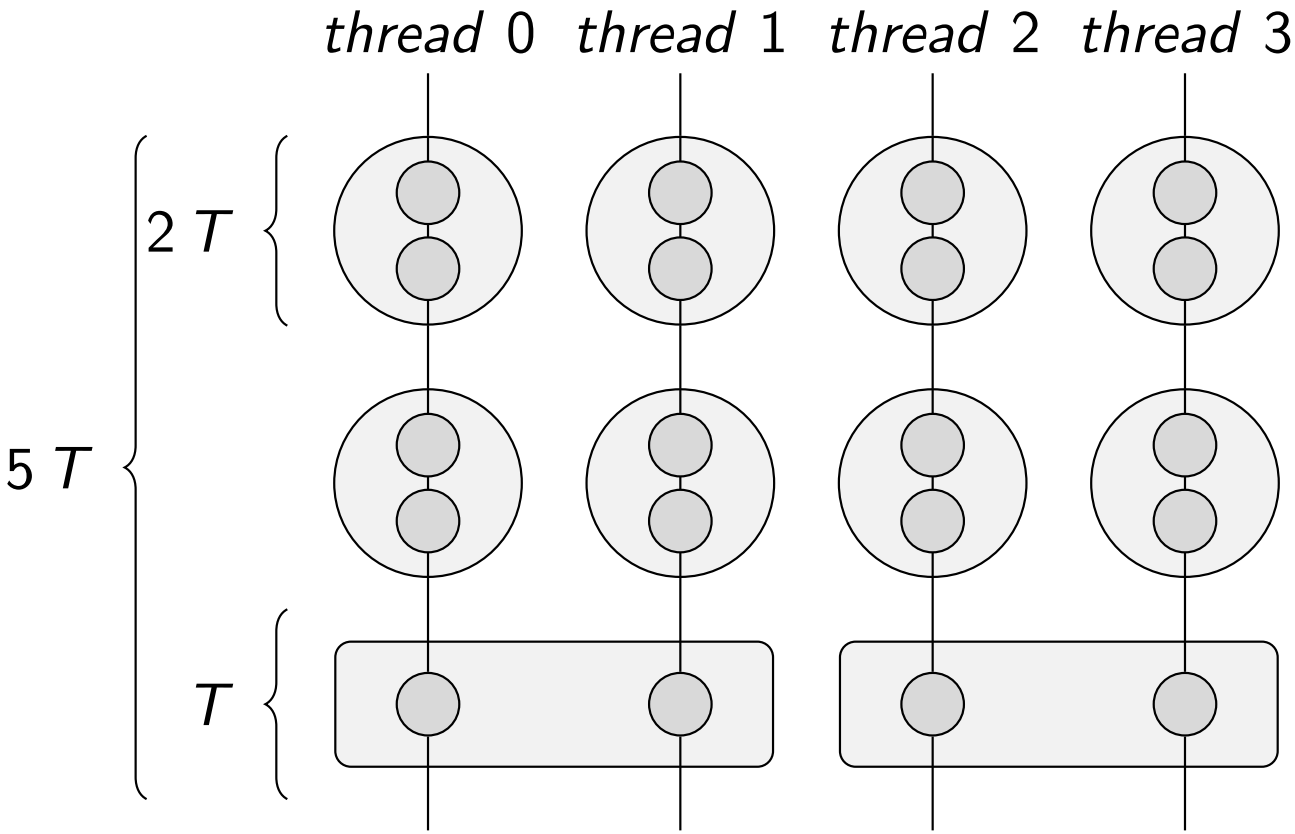
Répartition équilibrée multi-niveaux



10×2 tâches $= 2 \times 2$ tâches/*thread* $\times 4$ *threads* $+ 2 \times 2$ tâches

$\Longrightarrow$ accélération idéale : $3.\overline{3}$

Répartition équilibrée multi-niveaux



10×2 tâches $= (2 \times 2 + 1)$ tâches/*thread* $\times$ 4 *threads*

$\Longrightarrow$ accélération idéale : 4

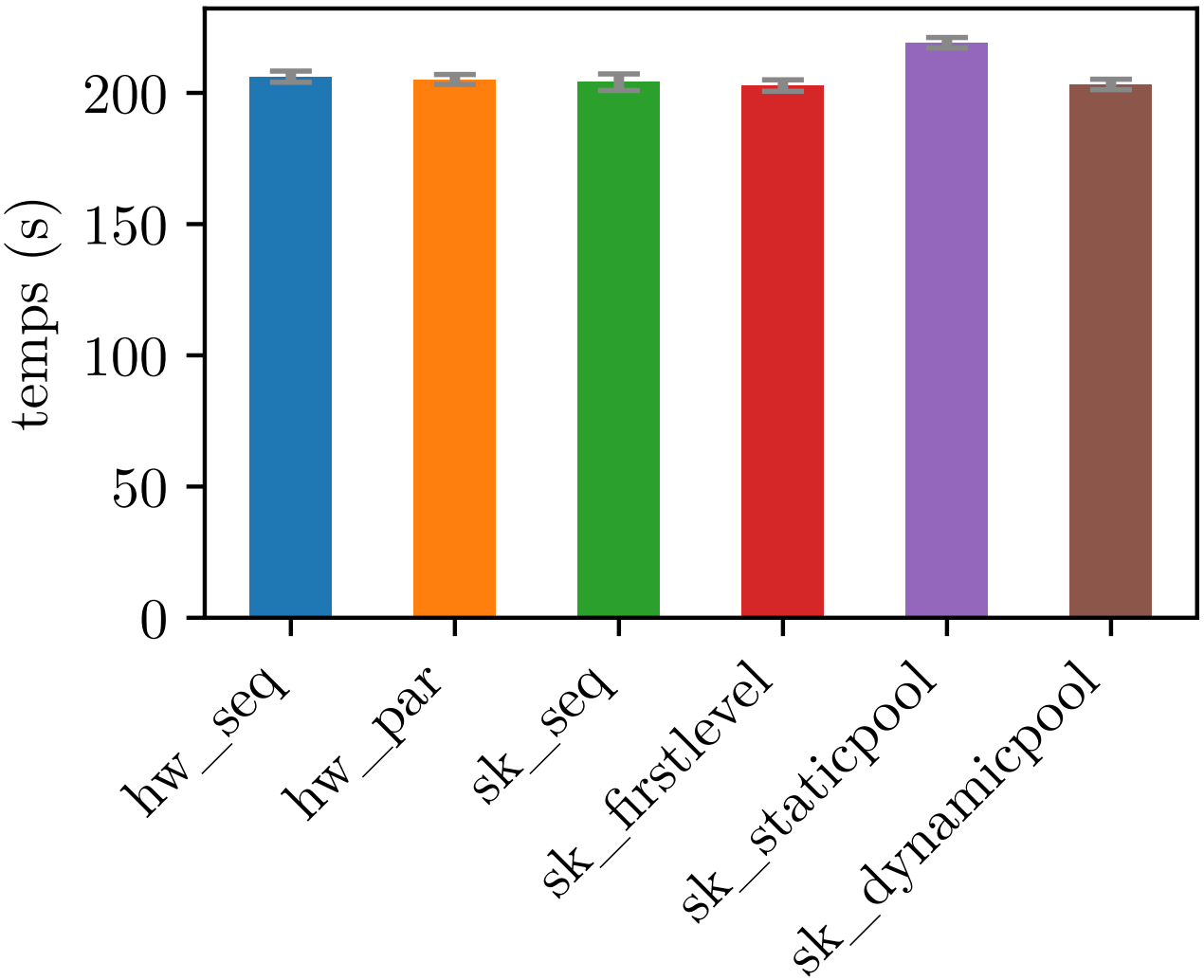
Performances

Contexte matériel et logiciel :

- Intel Xeon E7-8890 v3 – 2,5 GHz
- 18 cœurs
- GCC 8.2.0 – -O2
- Résolution de GRASP×ELS (194 sommets)
- moyennes sur 20 exécutions
- exécutions répétables

Performances

Temps moyen (exécutions séquentielles)

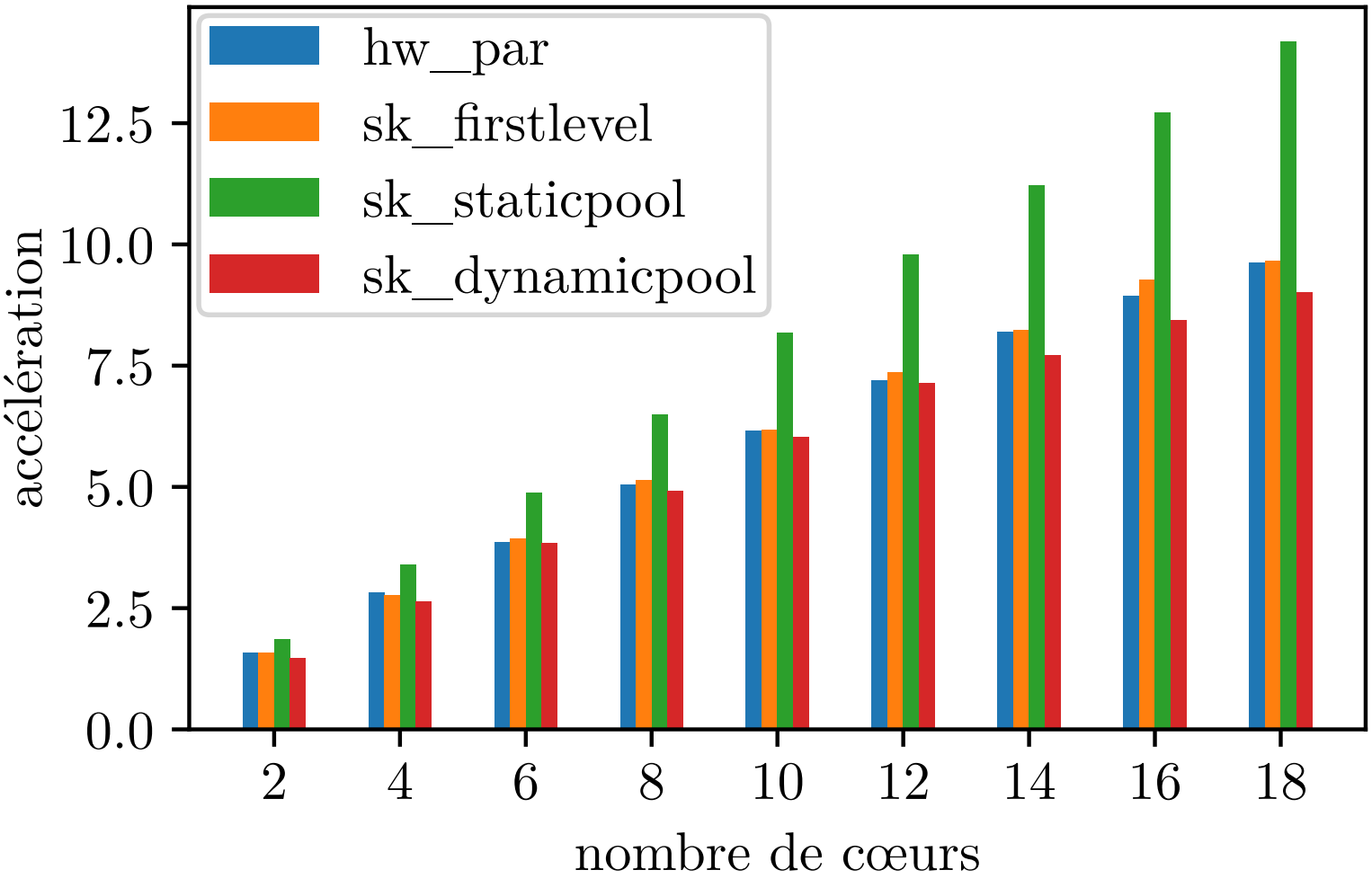


hw_... « à la main »

sk_... avec squelettes

Performances

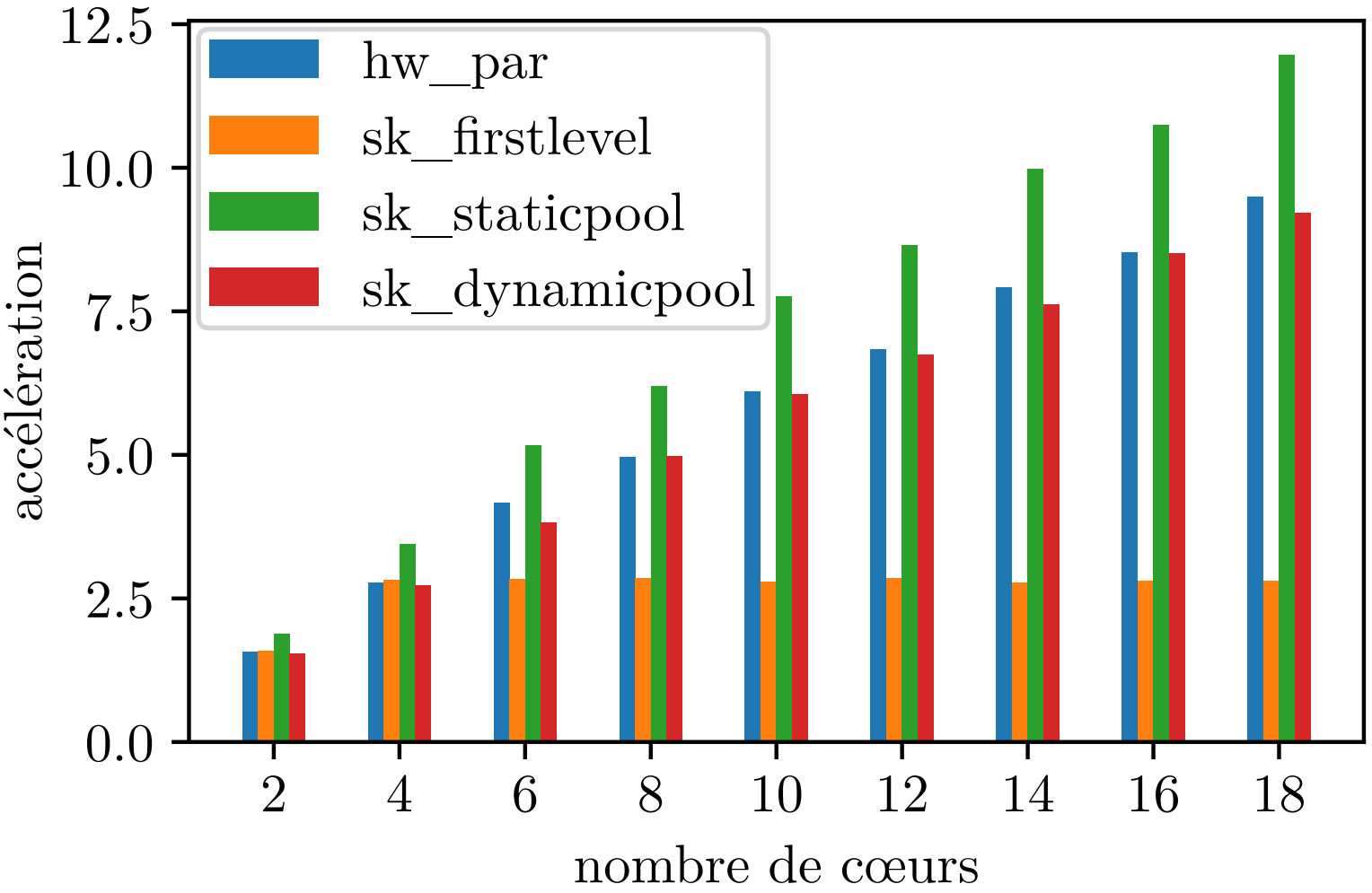
Temps moyen (exécutions parallèles)



Nombre d'itérations du GRASP : 20

Performances

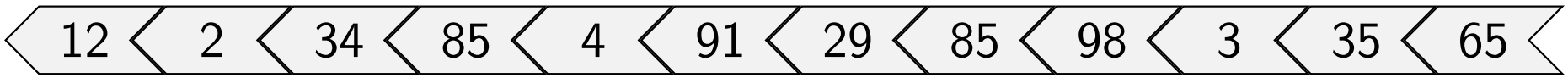
Accélération (exécutions parallèles)



Nombre d'itérations du GRASP : 4

Problème

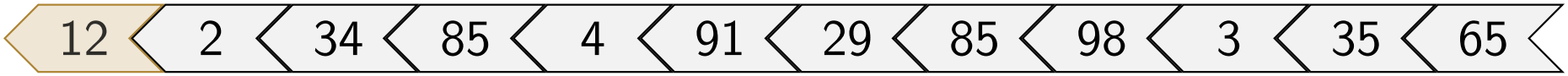
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow$

Problème

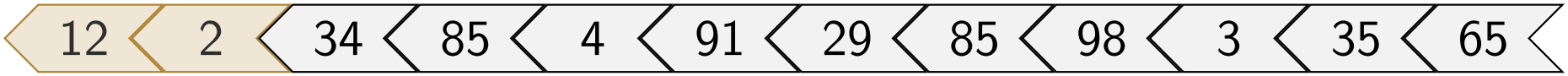
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12$

Problème

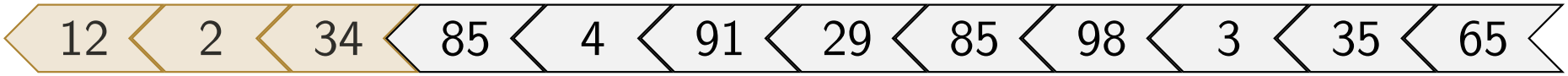
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 2$

Problème

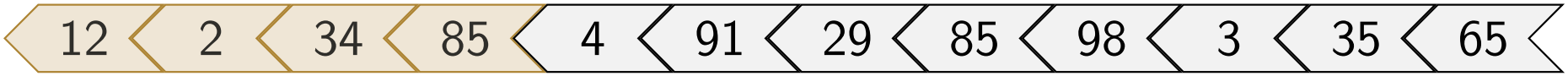
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 2,\ 34$

Problème

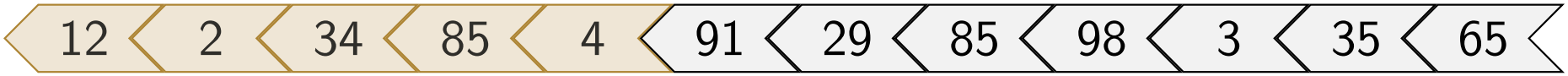
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 2,\ 34,\ 85$

Problème

Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 2,\ 34,\ 85,\ 4$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 2,\ 34,\ 85,\ 4,\ 91$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98, 3$

Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98, 3, 35$

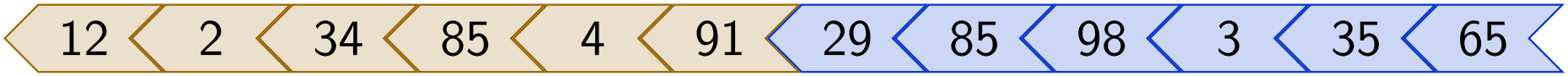
Problème

Exécution séquentielle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98, 3, 35, 65$

Problème

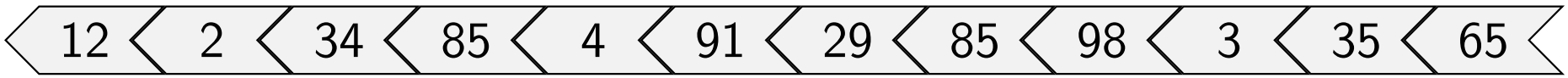
Exécution séquentielle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98, 3, 35, 65$

Problème

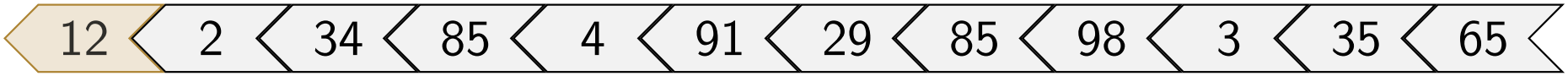
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow$

Problème

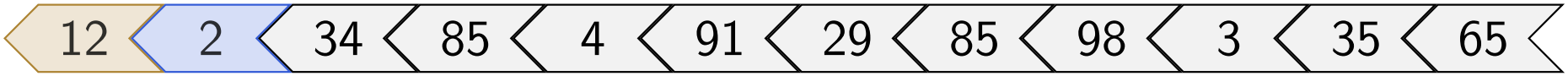
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12$

Problème

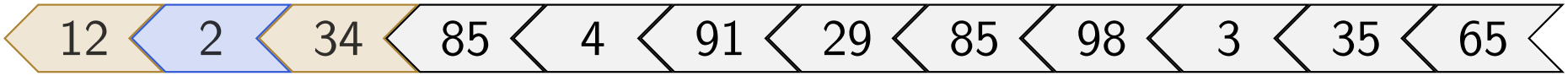
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2$

Problème

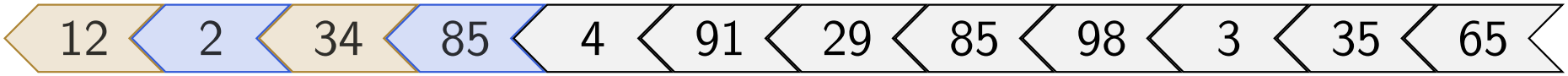
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2$

Problème

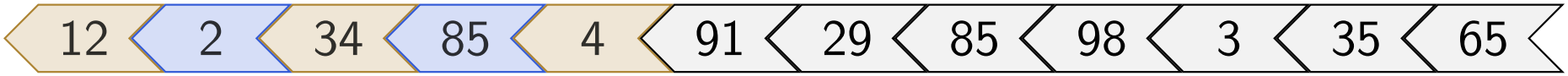
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85$

Problème

Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85$

Problème

Exécution parallèle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91$

Problème

Exécution parallèle

- $\frac{\text{tâche 0}}{\textit{thread 0}} \rightarrow 12, 34, 4, 29$
- $\frac{\text{tâche 1}}{\textit{thread 0}} \rightarrow 2, 85, 91$

Problème

Exécution parallèle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4,\ 29$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91,\ 85$

Problème

Exécution parallèle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4,\ 29,\ 98$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91,\ 85$

Problème

Exécution parallèle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4,\ 29,\ 98$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91,\ 85,\ 3$

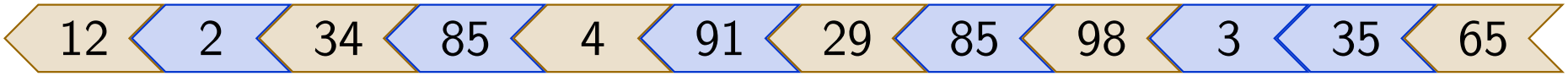
Problème

Exécution parallèle

- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4,\ 29,\ 98$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91,\ 85,\ 3,\ 35$

Problème

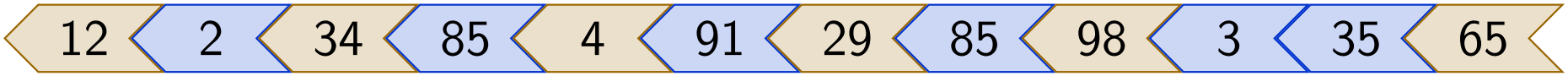
Exécution parallèle



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12,\ 34,\ 4,\ 29,\ 98,\ 65$
- $\frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 2,\ 85,\ 91,\ 85,\ 3,\ 35$

Problème

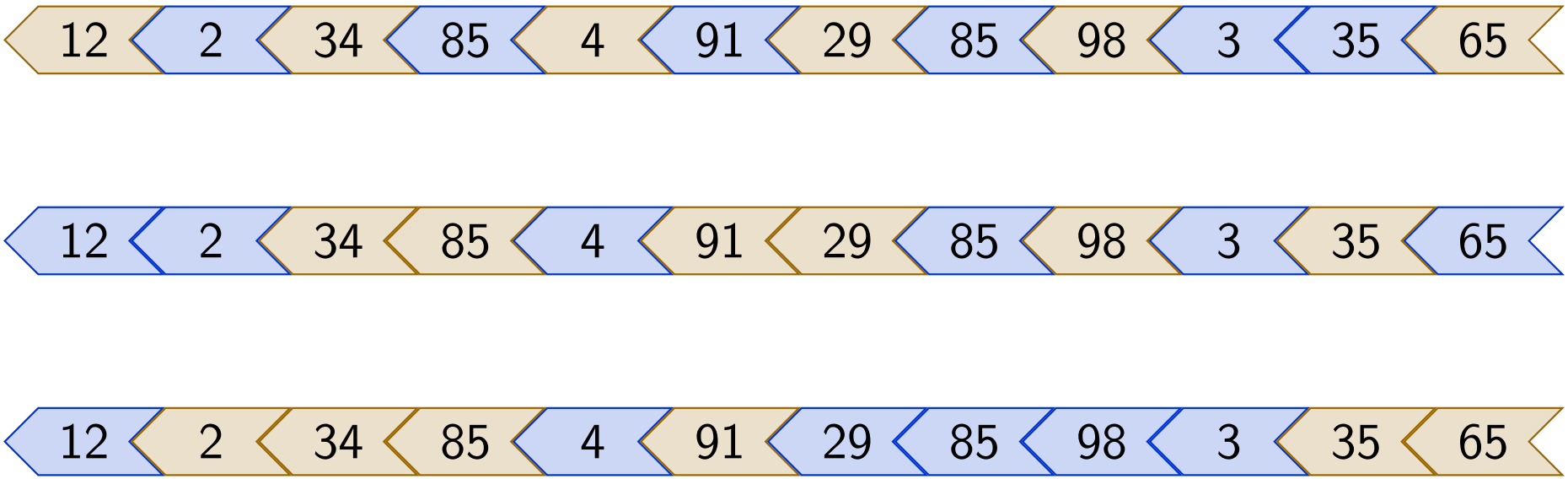
Exécution parallèle



- $\frac{\text{tâche 0}}{\textit{thread 0}} \rightarrow$ 12, 34, 4, 29, 98, 65 *ou* 34, 85, 91, 29, 98, 35
- $\frac{\text{tâche 1}}{\textit{thread 0}} \rightarrow$ 2, 85, 91, 85, 3, 35 *ou* 12, 2, 4, 85, 3, 65

Problème

Exécution parallèle



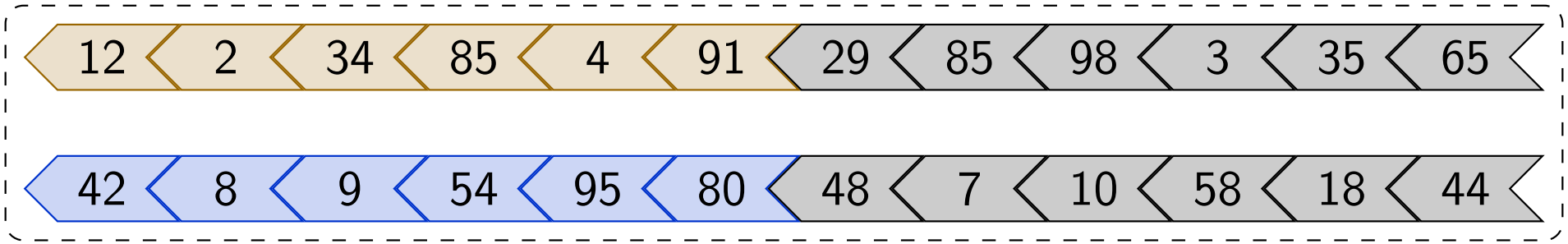
- $\frac{\text{tâche 0}}{\textit{thread 0}} \rightarrow$ 12, 34, 4, 29, 98, 65 *ou* 34, 85, 91, 29, 98, 35 *ou* ...
- $\frac{\text{tâche 1}}{\textit{thread 0}} \rightarrow$ 2, 85, 91, 85, 3, 35 *ou* 12, 2, 4, 85, 3, 65 *ou* ...

Solutions possibles

- un PRNG global $\Longrightarrow$ aucune répétabilité
- un PRNG par *thread* $\Longrightarrow$ répétabilité partielle
- un PRNG par tâche parallèle $\Longrightarrow$ répétabilité totale

Solutions possibles

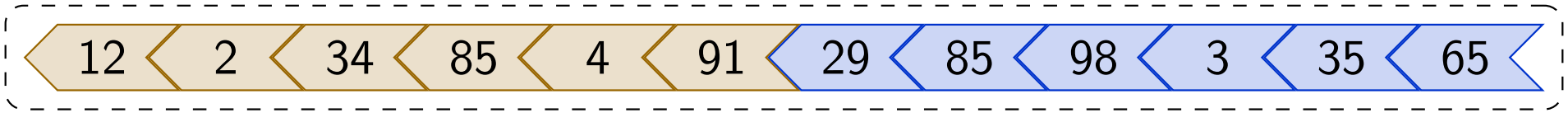
→ Un PRNG par *thread*



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 1} \rightarrow 42, 8, 9, 54, 95, 80$

Solutions possibles

→ Un PRNG par *thread*



- $\frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91 \quad = \quad \frac{\text{tâche } 0}{\textit{thread } 0} \rightarrow 12, 2, 34, 85, 4, 91$
- $\frac{\text{tâche } 1}{\textit{thread } 1} \rightarrow 42, 8, 9, 54, 95, 80 \quad \neq \quad \frac{\text{tâche } 1}{\textit{thread } 0} \rightarrow 29, 85, 98, 3, 35, 65$

Solutions possibles

→ Un PRNG par tâche parallèle

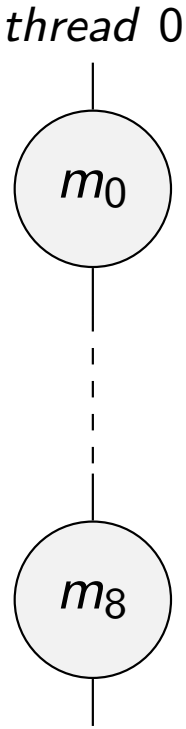
Pré-requis

- Système de liens (`RNG`)
- Identifier les tâches parallèles
  - parcours de l'arbre du squelette algorithmique
  - connaissance de la nature des motifs d'exécution

Inconvénient

- Beaucoup de PRNG créés
  - → réduction grâce à la connaissance des politiques d'exécution

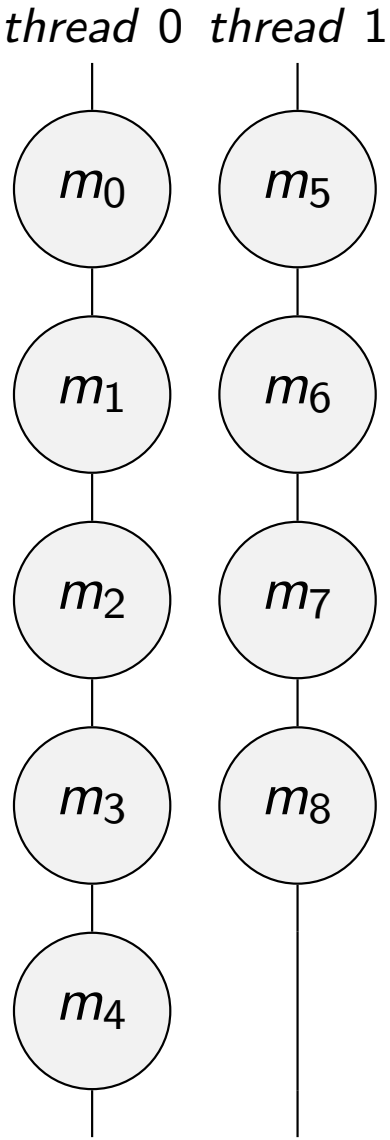
Optimiser le nombre de PRNG



Contraintes $\#threads \implies$ moins de PRNG

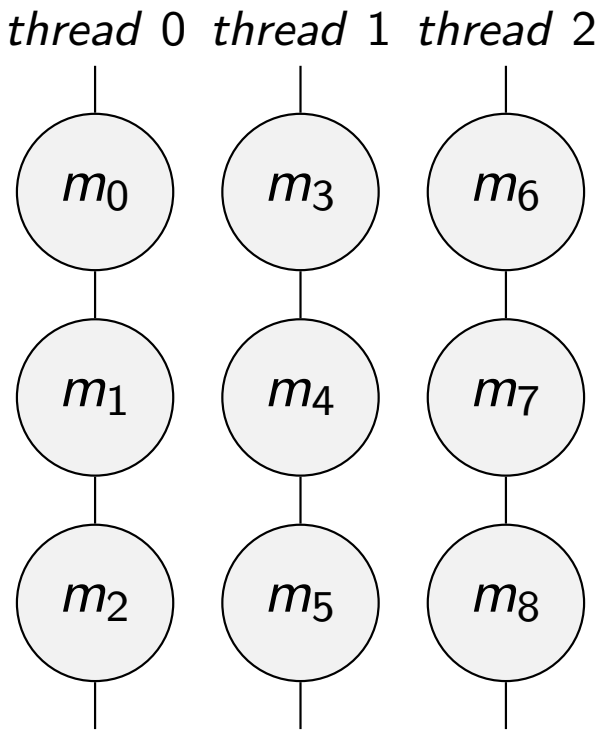
$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$

Optimiser le nombre de PRNG



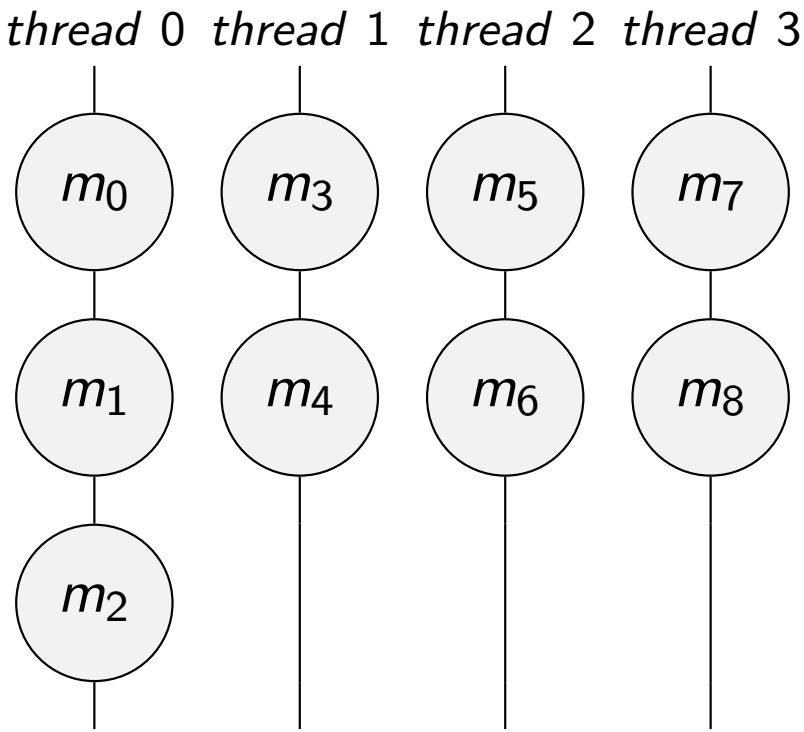
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$

Optimiser le nombre de PRNG



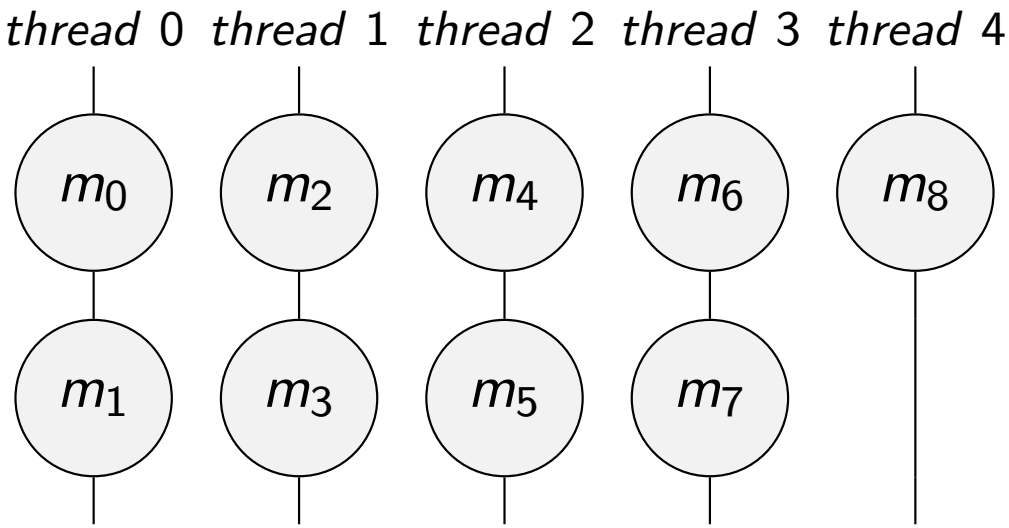
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$

Optimiser le nombre de PRNG



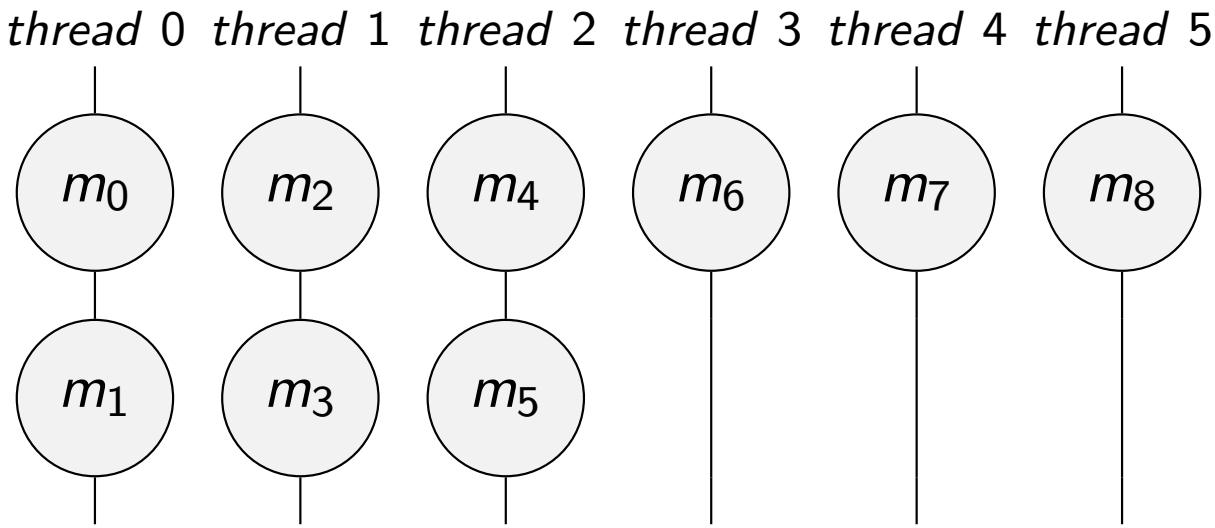
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG



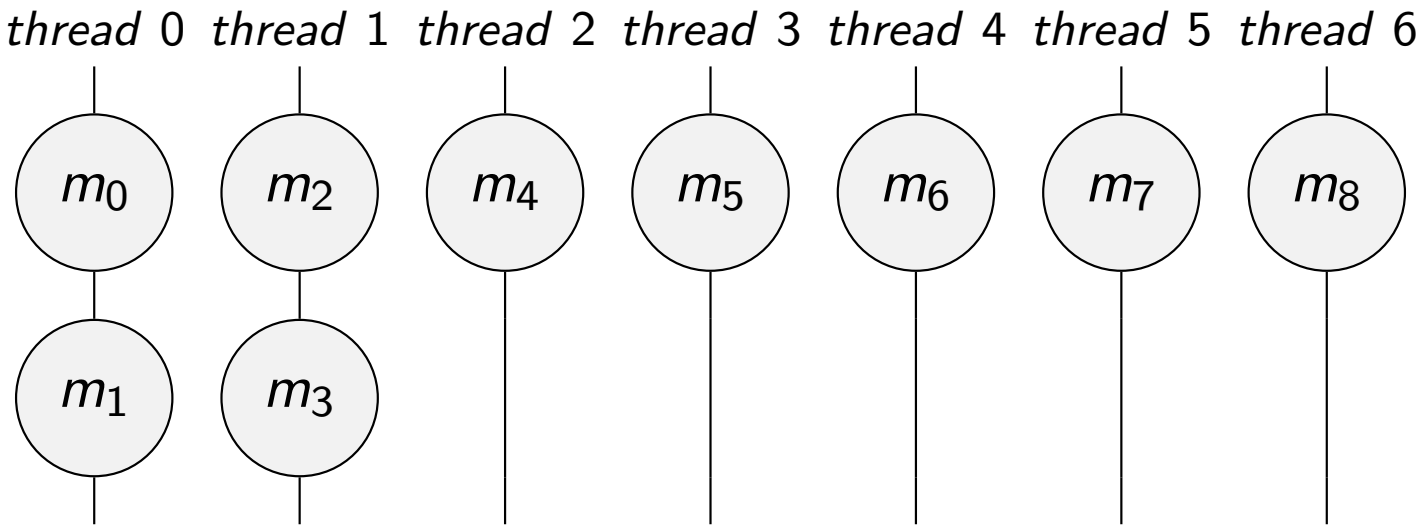
$$S_5 = \{\{m_0, m_1\}, \{m_2, m_3\}, \{m_4, m_5\}, \{m_6, m_7\}, \{m_8\}\}$$

Optimiser le nombre de PRNG



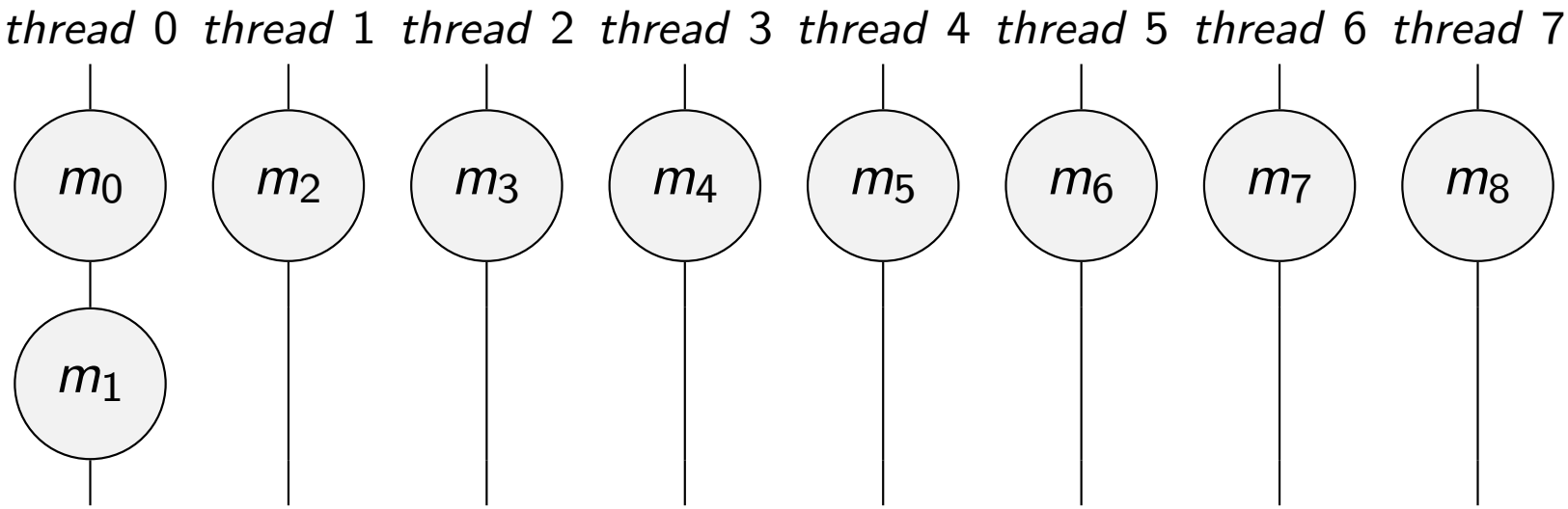
$$S_6 = \{\{m_0, m_1\}, \{m_2, m_3\}, \{m_4, m_5\}, \{m_6\}, \{m_7\}, \{m_8\}\}$$

Optimiser le nombre de PRNG



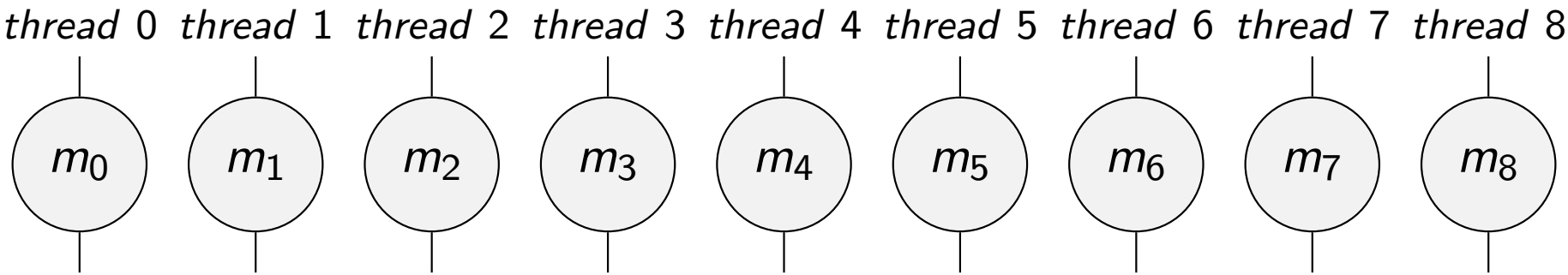
$$S_7 = \{\{m_0, m_1\}, \{m_2, m_3\}, \{m_4\}, \{m_5\}, \{m_6\}, \{m_7\}, \{m_8\}\}$$

Optimiser le nombre de PRNG



$$S_8 = \{\{m_0, m_1\}, \{m_2\}, \{m_3\}, \{m_4\}, \{m_5\}, \{m_6\}, \{m_7\}, \{m_8\}\}$$

Optimiser le nombre de PRNG



$$S_9 = \{\{m_0\}, \{m_1\}, \{m_2\}, \{m_3\}, \{m_4\}, \{m_5\}, \{m_6\}, \{m_7\}, \{m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

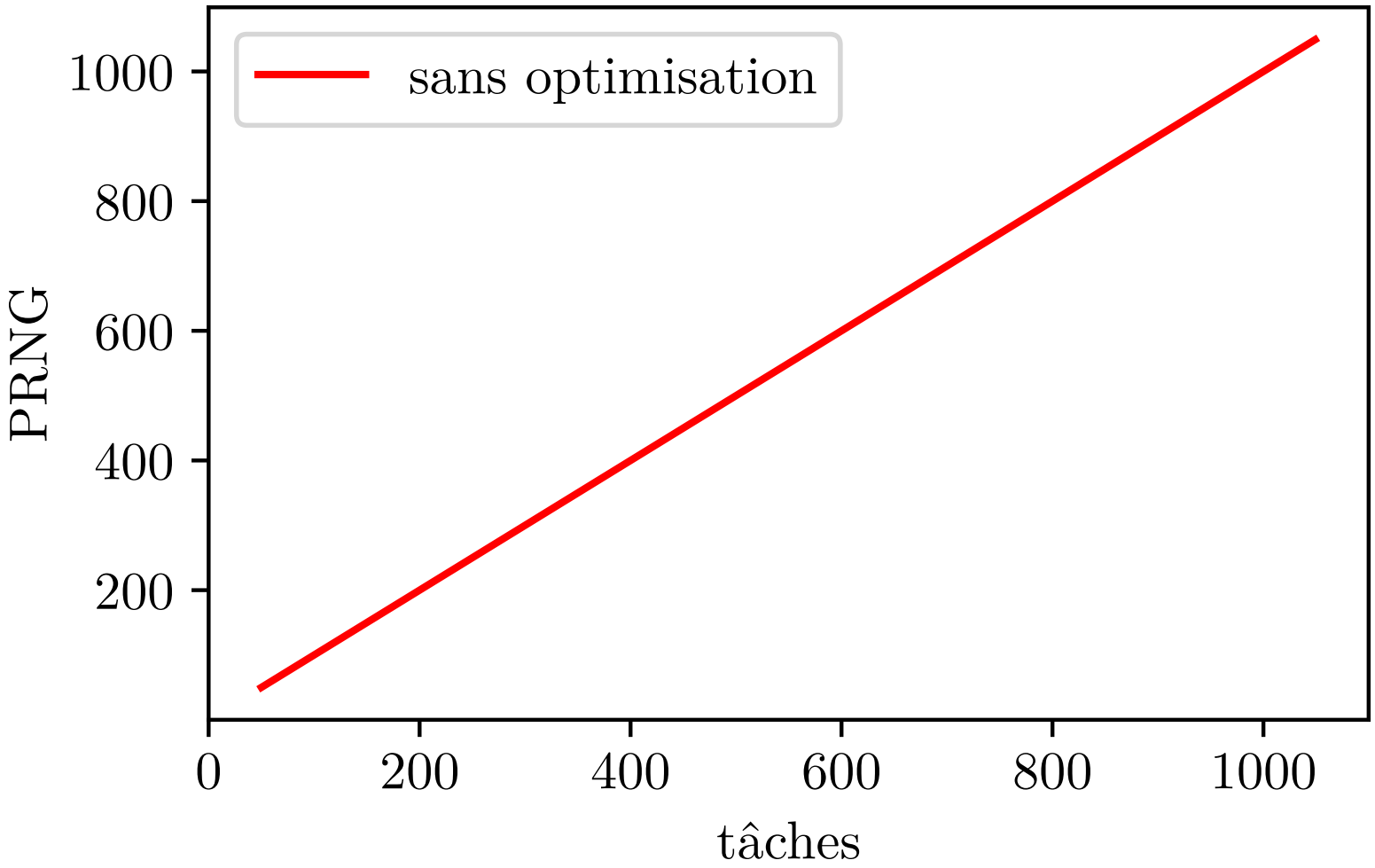
$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

Optimiser le nombre de PRNG

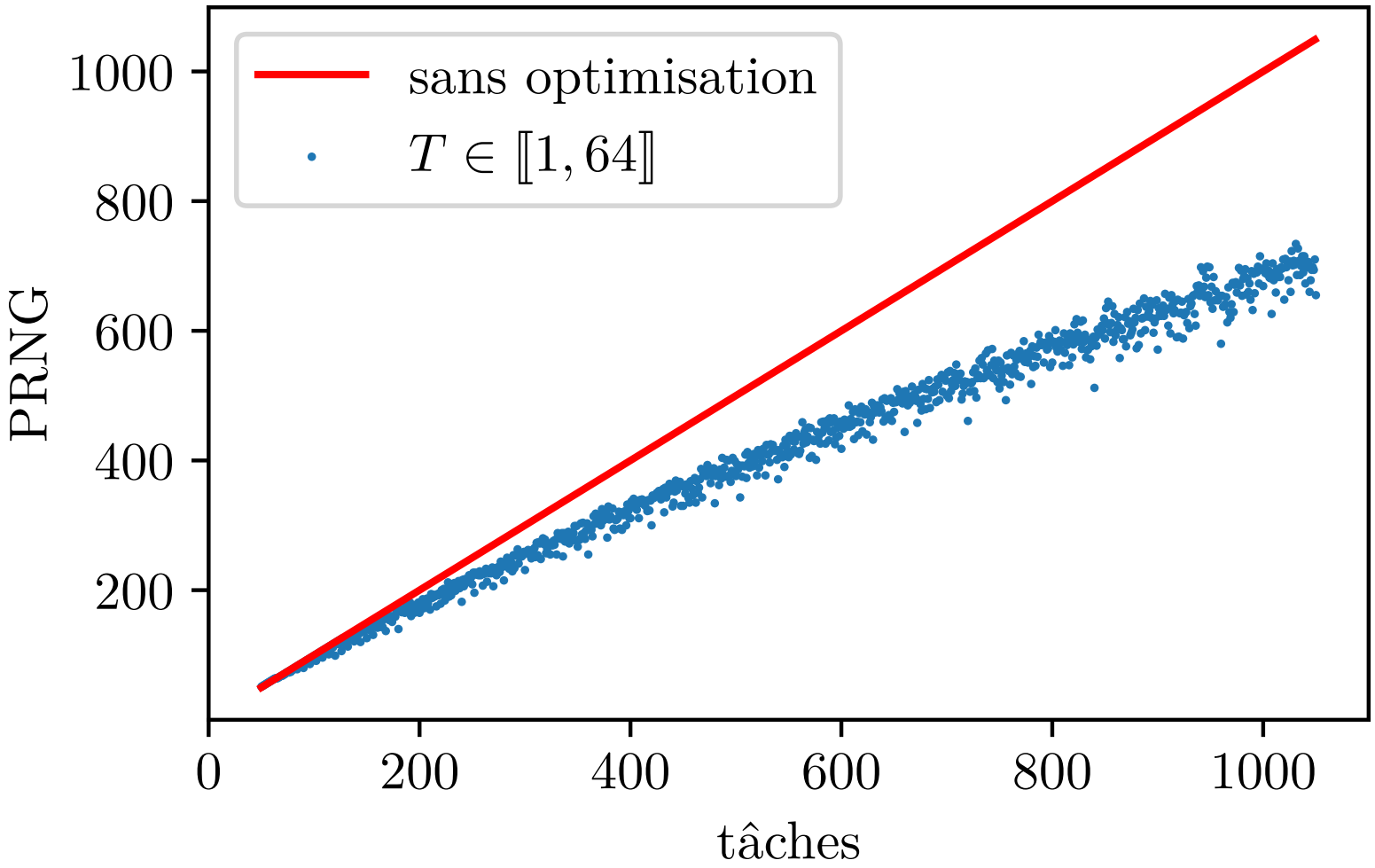
$$S_1 = \{\{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}\}$$
$$S_2 = \{\{m_0, m_1, m_2, m_3, m_4\}, \{m_5, m_6, m_7, m_8\}\}$$
$$S_3 = \{\{m_0, m_1, m_2\}, \{m_3, m_4, m_5\}, \{m_6, m_7, m_8\}\}$$
$$S_4 = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5, m_6\}, \{m_7, m_8\}\}$$

$$S = \{\{m_0, m_1, m_2\}, \{m_3, m_4\}, \{m_5\}, \{m_6\}, \{m_7, m_8\}\}$$

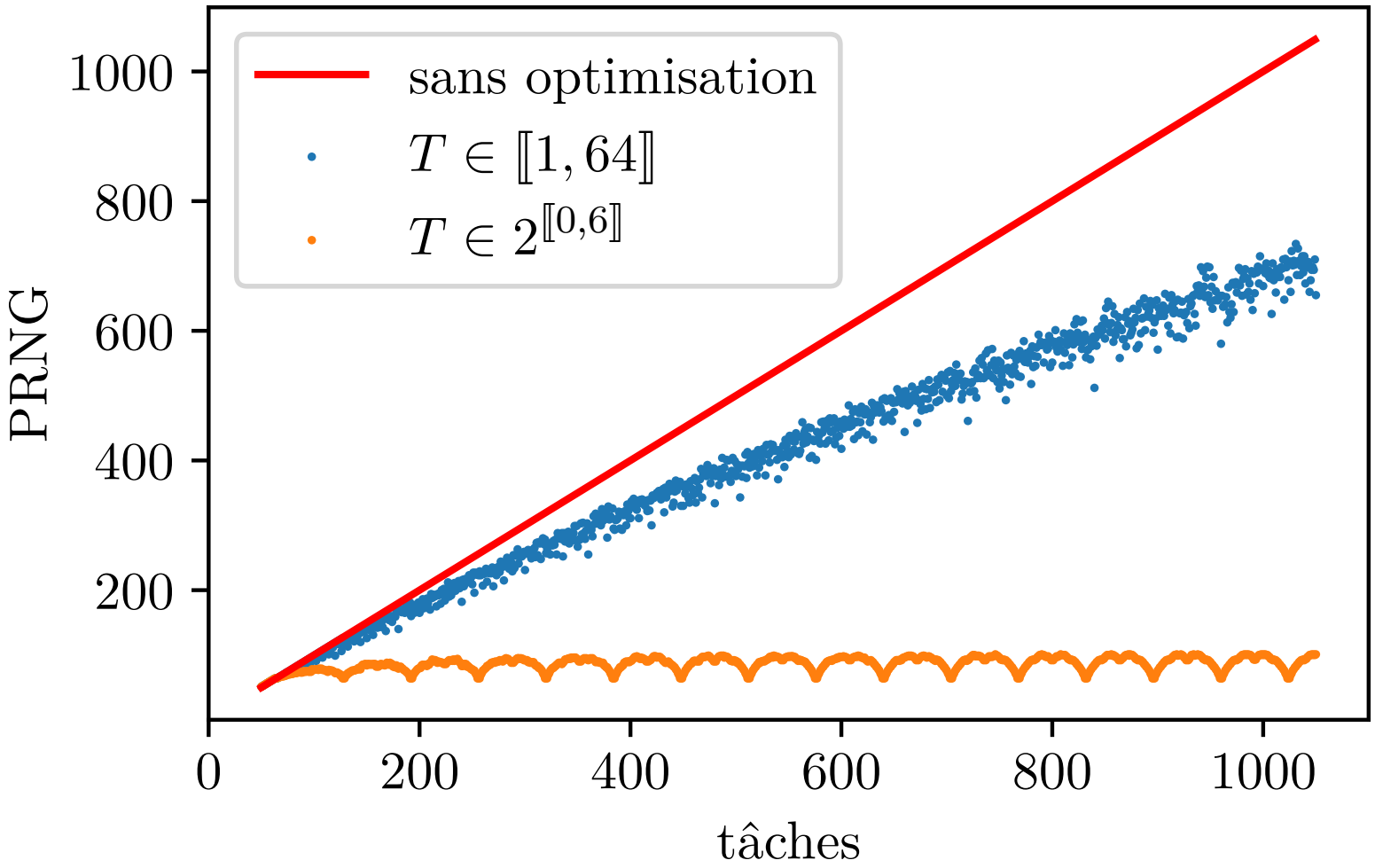
Nombre de PRNG nécessaires



Nombre de PRNG nécessaires



Nombre de PRNG nécessaires



Plusieurs niveaux d'utilisation :

- conception de motifs d'exécution
- implémentation de politiques d'exécution
- définition de squelettes (bibliothèques spécialisées)
- utilisation finale d'un squelette existant

Répétabilité :

- indépendante du nombre de *threads*
- optimisée en fonction de la politique d'exécution

Plan

Besoin

Temps d'exécution concentré dans les boucles

$\Longrightarrow$ parallélisation des boucles

Besoin

Temps d'exécution concentré dans les boucles

$\Longrightarrow$ parallélisation des boucles

```
for(int i = 0; i < N; ++i)
  a[i] = a[i] + 1;
```

Besoin

Temps d'exécution concentré dans les boucles

$\Longrightarrow$ parallélisation des boucles

```
for(int i = 0; i < N; ++i)
  a[i] = a[i] + 1;
```

```
for(int i = 1; i < N; ++i)
  a[i] = a[i-1];
```

Besoin

Temps d'exécution concentré dans les boucles

$\Longrightarrow$ parallélisation des boucles

```
for(int i = 0; i < N; ++i)
  a[i] = a[i] + 1;
```

```
for(int i = 1; i < N; ++i)
  a[i] = a[i-1];
```

```
for(int i = 1; i < N; i += 2)
  a[i] = a[i-1];
```

Besoin

Temps d'exécution concentré dans les boucles

$\Longrightarrow$ parallélisation des boucles

```
for(int i = 0; i < N; ++i)
  a[i] = a[i] + 1;
```

```
for(int i = 1; i < N; ++i)
  a[i] = a[i-1];
```

```
for(int i = 1; i < N; i += 2)
  a[i] = a[i-1];
```

Objectif de la bibliothèque :

- analyse automatique durant la compilation
- génération automatique

Analyse

Conditions de Bernstein

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  c[i] = c[i+1] - d[i];
  b[i] = b[i] + i;
  d[i] = std::pow(c[i], e[i]);
  f[i*i] = 2 * f[i*i];
}
```

Analyse

Conditions de Bernstein

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  c[i] = c[i+1] - d[i];
  b[i] = b[i] + i;
  d[i] = std::pow(c[i], e[i]);
  f[i*i] = 2 * f[i*i];
}
```

Indépendance de deux instructions x et y selon Bernstein

$$W_y \cap R_x = \emptyset$$
$$R_y \cap W_x = \emptyset$$
$$W_x \cap W_y = \emptyset$$

Analyse

Conditions de Bernstein

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  c[i] = c[i+1] - d[i];
  b[i] = b[i] + i;
  d[i] = std::pow(c[i], e[i]);
  f[i*i] = 2 * f[i*i];
}
```

Indépendance de deux instructions x et y selon Bernstein

$$W_y \cap R_x = \{b\}$$
$$R_y \cap W_x = \emptyset$$
$$W_x \cap W_y = \emptyset$$

Analyse

Conditions de Bernstein

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  c[i] = c[i+1] - d[i];
  b[i] = b[i] + i;
  d[i] = std::pow(c[i], e[i]);
  f[i*i] = 2 * f[i*i];
}
```

Indépendance de deux instructions x et y selon Bernstein

$$W_4 \cap R_2 = \{d\}$$

$$R_4 \cap W_2 = \{c\}$$

$$W_2 \cap W_4 = \emptyset$$

Analyse

Conditions de Bernstein

```
for(int i = 0; i < n; ++i) {
    a[i] = a[i] * b[i];
    c[i] = c[i+1] - d[i];
    b[i] = b[i] + i;
    d[i] = std::pow(c[i], e[i]);
    f[i*i] = 2 * f[i*i];
}
```

→ analyse des variables

Analyse

Identification des boucles parallélisables

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  b[i] = b[i] + i;
}

for(int i = 0; i < n; ++i) {
  c[i] = c[i+1] - d[i];
  d[i] = std::pow(c[i], e[i]);
}

for(int i = 0; i < n; ++i)
  f[i*i] = 2 * f[i*i];
```

Analyse

Identification des boucles parallélisables

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  b[i] = b[i] + i;
}

for(int i = 0; i < n; ++i) {
  c[i] = c[i+1] - d[i];
  d[i] = std::pow(c[i], e[i]);
}

for(int i = 0; i < n; ++i)
  f[i*i] = 2 * f[i*i];
```

→ analyse des fonctions d'indice

Analyse

Résultat

```
#pragma omp parallel for
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  b[i] = b[i] + i;
  f[i*i] = 2 * f[i*i];
}

for(int i = 0; i < n; ++i) {
  c[i] = c[i+1] - d[i];
  d[i] = std::pow(c[i], e[i]);
}
```

Utilisation

Boucle originale

```
for(int i = 0; i < n; ++i) {
  a[i] = a[i] * b[i];
  c[i] = c[i+1] - d[i];
  b[i] = b[i] + i;
  d[i] = std::pow(c[i], e[i]);
  f[i*i] = 2 * f[i*i];
}
```

Boucle active

```
parallelFor(Range{0, n},
  a[i] = a[i] * b[i],
  c[i] = c[i+ctv<1>] - d[i],
  b[i] = b[i] + i,
  d[i] = pow(c[i], e[i])
  f[i*i] = 2 * f[i*i]
);
```

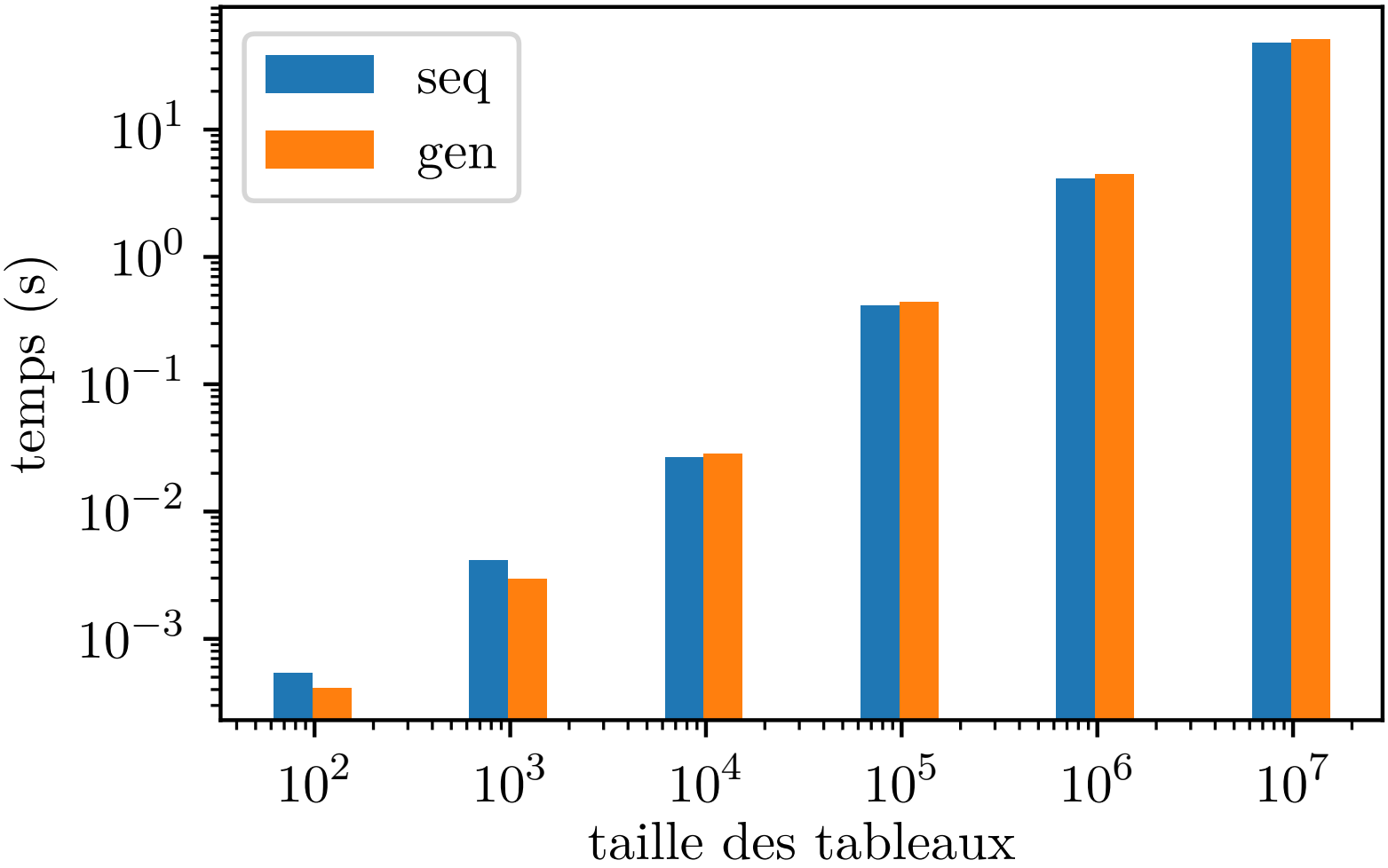
Performances

Contexte matériel et logiciel :

- Intel Xeon E7-8890 v3 – 2,5 GHz
- 18 cœurs
- GCC 8.2.0 – -O2
- moyennes sur 20 exécutions
- exécutions répétables

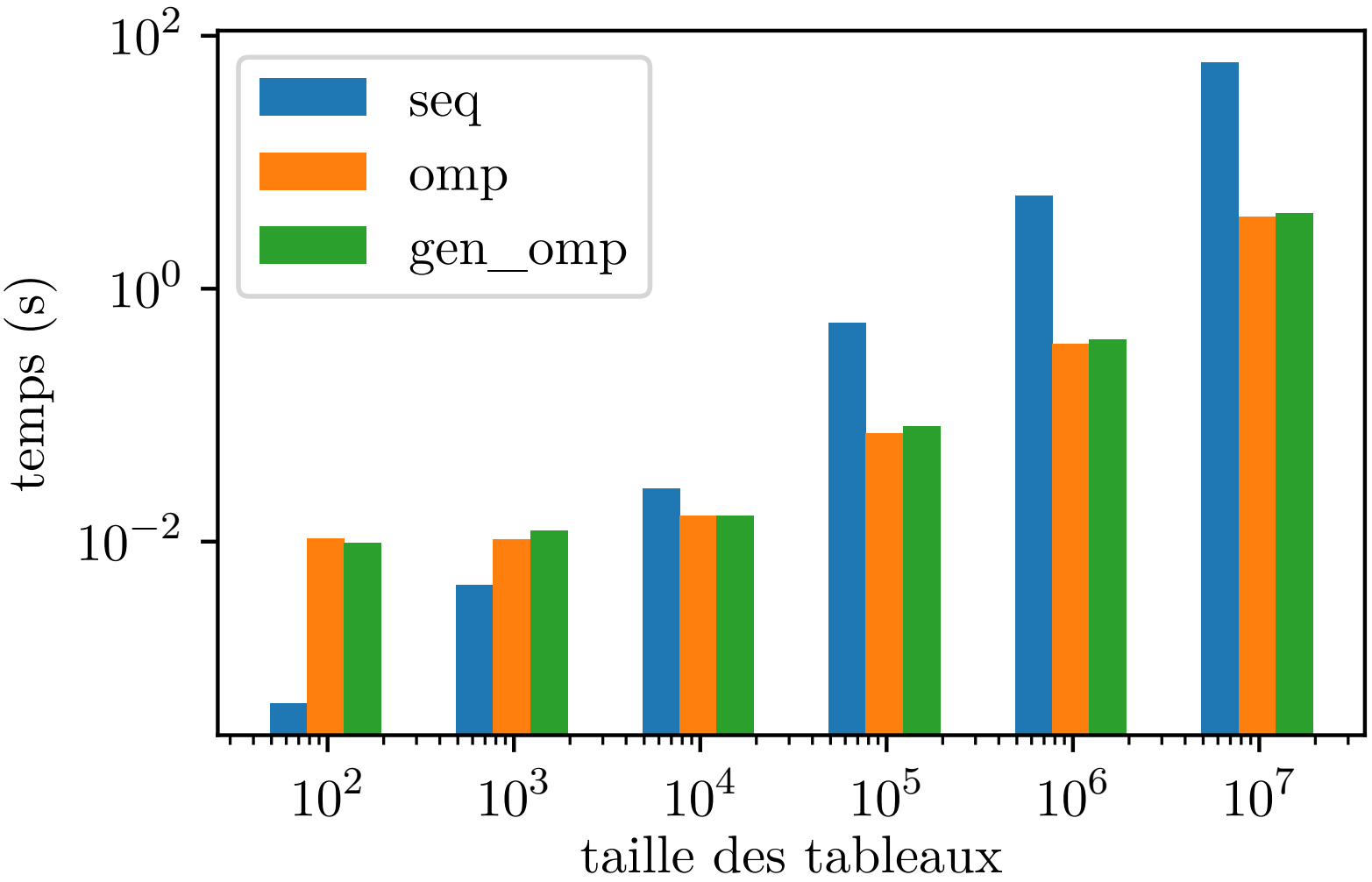
Performances

Exécutions séquentielles

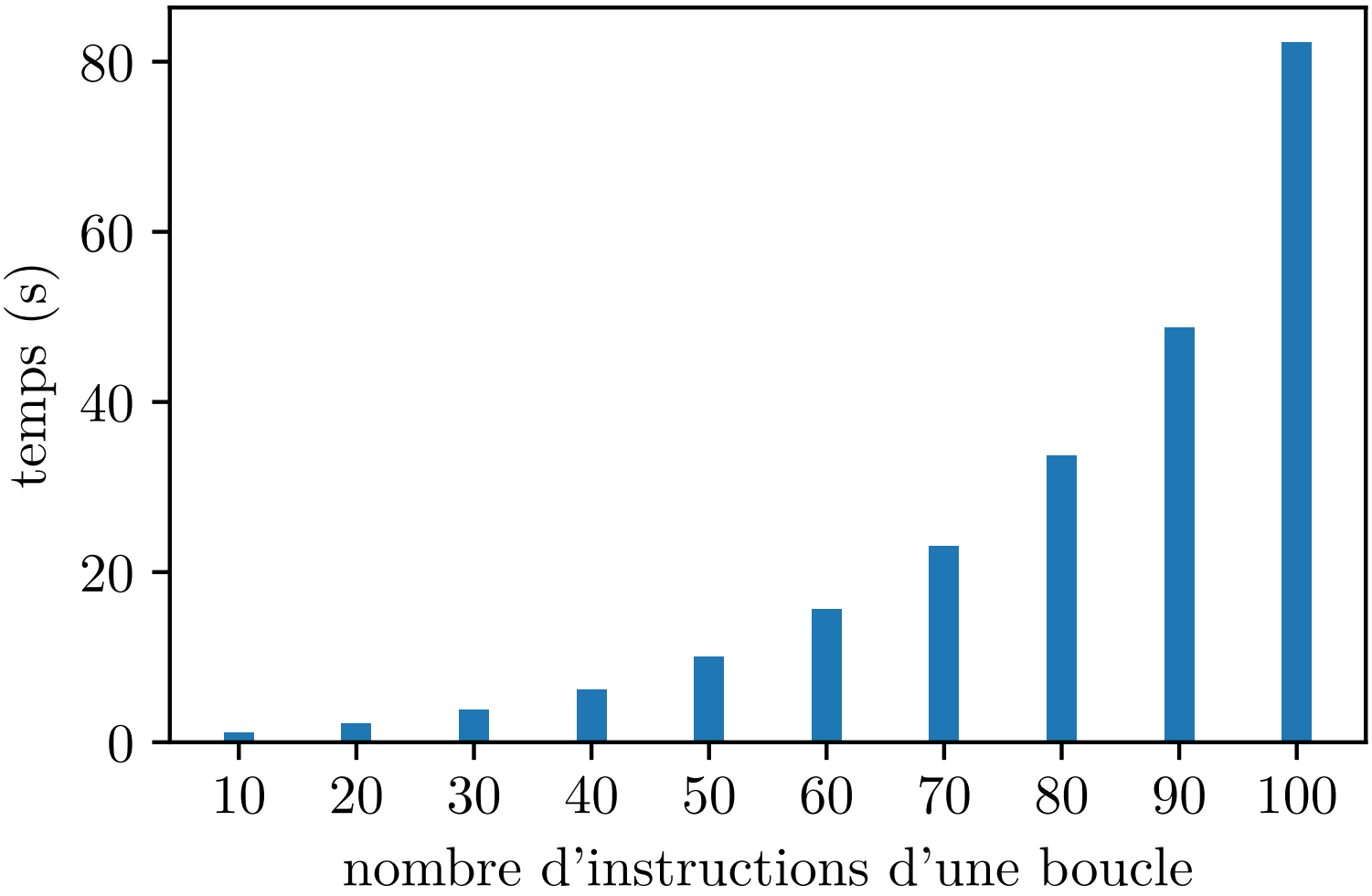


Performances

Exécutions parallèles



Performances – compilation



Bilan

2 bibliothèques actives de parallélisation :

- abstraction sans surcoût à l'exécution
- parallélisation assistée ou automatique

- Bibliothèque active de parallélisation assistée :
  - squelettes algorithmiques
  - choix de la politique d'exécution
  - répétabilité garantie automatiquement
  - quelques squelettes pour la recherche opérationnelle
- Bibliothèque active de parallélisation automatique de boucles :
  - analyse à la compilation de l'AST
    - ...des variables pour les dépendances
    - ...des fonctions d'indice pour la parallélisabilité

Limites et perspectives

Bibliothèque de parallélisation assistée :

- nouveaux motifs :
  - *pipeline*
  - *divide and conquer*
  - ...
- politique d'exécution équilibrée pondérée

Bibliothèque de parallélisation automatique de boucles :

- généralisation de l'analyse des fonctions d'indice
- transformation d'un code non parallélisable